UNIVERSIDADE DE LISBOA
FACULDADE DE CIÊNCIAS
DEPARTAMENTO DE INFORMÁTICA

CONTROLO DE DISPOSITIVOS MÉDICOS EM TEMPO REAL

Tiago Manuel Cardoso Louro dos Reis

MESTRADO EM ENGENHARIA INFORMÁTICA
Especialização em Sistemas de Informação

Versão Pública

Trabalho de Projeto orientado por:
Prof. Doutor José Manuel da Silva Cecílio
e por Doutor Paulo Jorge Paiva de Sousa

2020

Resumo

O presente documento relata as atividades, experiências e conhecimentos obtidos durante a realização do Projeto em Engenharia Informática (PEI), integrado no 2º e último ano do Mestrado em Engenharia Informática da Faculdade de Ciências da Universidade de Lisboa (FCUL). Este projeto foi elaborado na instituição Maxdata Software, SA, sediada no Carregado, Portugal e teve como resultado um novo produto que integrará o leque de soluções para a área da saúde desta empresa.

Neste projeto foi desenvolvido o Clinidata® Instruments: um sistema distribuído de comunicação e controlo de dispositivos médicos em tempo real que funciona com a aplicação Clinidata®, o principal produto comercializado pela Maxdata Software, SA.

O controlo de diferentes dispositivos médicos é possível através do uso de *interfaces* de comunicação, tipicamente disponibilizadas pelos fabricantes dos mesmos. Muitas vezes estes fabricantes optam por criar a sua interface proprietária, ainda assim, existem muitos factores comuns de dispositivo para dispositivo, sendo possível criar um software comum onde se adicionam especificações para cada dispositivo.

O *software* desenvolvido situa-se 'entre' os dispositivos médicos e o Clinidata®, sendo considerado um *middleware* que estabelece uma ligação bidirecional entre estes dois componentes e faz o processamento de dados, oferecendo mecanismos de tolerância a falhas que são fundamentais num sistema distribuído, especialmente em sistemas críticos como os que são utilizados na área da saúde.

O Clinidata® Instruments encontra-se a ser utilizado em dois laboratórios: no Centro de Testes à COVID-19 da Faculdade de Ciências da Universidade de Lisboa e no Laboratório de Patologia Clínica do IPO de Lisboa.

Palavras-chave: Clinidata®, Dispositivos Médicos, Software, Saúde, Sistemas Distribuídos

Abstract

This document reports the activities, experiences and knowledge obtained during the Computer Engineering Project course, integrated in the 2nd and final year of the Master's Degree in Computer Engineering at the Faculty of Sciences of the University of Lisbon. This project was developed in the Maxdata Software, SA, Carregado, Portugal and resulted in a new product that will be integrated in the set of solutions for the health area of this company.

In this project the Clinidata® Instruments was developed: a distributed system of communication and control for medical devices. It operates in real time and integrates the Clinidata® application, the main product marketed by Maxdata Software, SA.

The control of different medical devices is possible through the use of communication interfaces, typically made available by their manufacturers. Often these manufacturers choose to create their own proprietary interface. However, there are many common factors from device to device, making possible to create common software where specific features are added for each device.

The developed software is located 'between' medical devices and Clinidata®, being considered a middleware that establishes a bidirectional connection between these two components and does the processing of data, offering fault tolerance mechanisms that are fundamental in a distributed system, especially in critical systems such as those used in healthcare.

Clinidata® Instruments is being used in two laboratories: COVID-19 Testing Centre of the Faculty of Sciences of the University of Lisbon and the Clinical Pathology Laboratory of the IPO in Lisbon.

Keywords: Clinidata®, Medical Devices, Software, Health, Distributed Systems

Conteúdo

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

O presente documento relata as atividades, experiências e conhecimentos obtidos durante a realização do Projeto em Engenharia Informática (PEI), integrado no 2º e último ano do Mestrado em Engenharia Informática da Faculdade de Ciências da Universidade de Lisboa (FCUL). Este projeto foi elaborado na instituição Maxdata Software, SA, sediada no Carregado, Portugal e teve como resultado um novo produto que integrará o leque de soluções para a área da saúde desta empresa.

A necessidade de sistemas de informação no setor laboratorial da área da saúde tem vindo a aumentar ao longo dos anos tornando-se fundamental, nos dias de hoje, a existência de uma solução de software que permita, fazer a aquisição, guardar, gerir e armazenar dados de um laboratório. Esta necessidade fez surgir uma nova categoria de software, os Sistemas de Informação Laboratorial, mais conhecidos pela sua denominação em inglês como Laboratory Information System (LIS).

Estas soluções vieram permitir uma automação de fases de todo o processo laboratorial trazendo para este vantagens como: maior rapidez, diminuição do fator erro humano, redução de custos, diminuição da carga de trabalho dos profissionais de saúde que participam neste processo, e a informatização deste processo.

Uma das fases do processo laboratorial é a fase analítica[6] em que são utilizados dispositivos médicos para realizar a análise do material recolhido em fases anteriores. Este dispositivos recebem como entrada uma amostra e a respetiva programação de análises a realizar nessa amostra e produzem resultados que podem ser recebidos através de uma interface disponibilizada pelos dispositivos. Em alguns casos, os fabricantes destes dispositivos disponibilizam um *software* que permite a sua programação e a recepção dos respectivos resultados produzidos. No entanto, esta abordagem tem algumas desvantagens, entre as quais, ser uma tarefa complexa e pouco eficiente controlar individualmente cada dispositivo sem qualquer tipo de automação e integração com um LIS.

No contexto do projeto descrito neste relatório foi desenvolvido um sistema genérico que permite a comunicação com dispositivos médicos através da especificação do funcionamento do comportamento de comunicação (protocolo) do dispositivo, tendo sido

desenvolvidas algumas dessas especificações que adicionaram ao sistema o suporte a 37 diferentes modelos de dispositivos médicos. Este sistema genérico apresenta um conjunto de funcionalidades inovadoras em relação à concorrência e que permite a integração deste tipo de dispositivos com o Clinidata® LIS, tornando possível controlo total destes dispositivos. Estes protocolos permitem lidar com a heterogeneidade dos dispositivos médicos analisadores existente em diferentes laboratórios.

Foi ainda escrito um artigo científico com o título 'Fault-Tolerant Architecture for Real-Time Control of Distributed Medical Devices', que se encontra anexado a este documento no Apêndice B. Este artigo descreve a arquitetura do sistema implementado com ênfase na tolerância a falhas e apresenta resultados de uma série de testes de realizados ao sistema com a finalidade de avaliar o desempenho. Este artigo foi primeiramente submetido para a conferência 'SafeComp 2020' [17], onde recebeu uma avaliação positiva e duas negativas, tendo sido rejeitado. No entanto, mais tarde, o artigo foi re-submetido e aceite na conferência 'ICARCV 2020' [11].

1.1 Objetivos

Os principais objetivos deste Projeto em Engenharia Informática foram:

- Definição de casos de uso e especificação completa de requisitos funcionais e não funcionais do sistema a ser desenvolvido;

- Desenho e conceção do sistema. Neste objetivo inclui-se o desenho da arquitetura do sistema, diagramas de desenvolvimento de software, o desenho do modelo de dados;

  De forma a aumentar a robustez do sistema, inclui-se também neste ponto a definição do funcionamento de uma cache para modo *offline*, a definição de mecanismos de gestão e controlo de protocolos e a definição de simulação de dispositivos para teste automático dos protocolos desenvolvidos;

- Implementação e respetiva avaliação de desempenho. Este objetivo divide-se em duas tarefas principais. A primeira corresponde a uma prova de conceito com três protocolos de dispositivos que utilizem mecanismos de comunicação diferentes (RS233, TCP/IP e Partilha de Ficheiros). Para cada um destes protocolos são desenvolvidos testes unitários automáticos e é avaliado o seu desempenho global de forma a identificar possíveis falhas. A segunda tarefa incide no desenvolvimento de outros protocolos para os restantes dispositivos médicos previstos e suportados pelo Clinidata®, utilizando a metodologia e resultados adquiridos na prova de conceito, seguindo as mesmas etapas de implementação, desenvolvimento de testes unitários automáticos de avaliação de desempenho.

1.2 Instituição de Acolhimento

A Maxdata Software, S.A. concebe, desenvolve, instala e mantém software para a área de saúde há 4 décadas. Este software é propriedade exclusiva da Maxdata e tem marca própria - Clinidata® - registada em toda a União Europeia. O software Clinidata® está presente nos maiores hospitais portugueses em várias áreas e laboratórios, nomeadamente, requisição eletrónica, patologia clínica, anatomia patológica, imunohemoterapia (banco de sangue e transfusões) e vigilância epidemiológica (controlo de infeções hospitalares). O software Clinidata® está também presente em laboratórios de referência nacionais e internacionais, incluindo o maior laboratório privado de anatomia patológica da Península Ibérica e 17 hospitais do Médio Oriente.

A experiência decorrente das centenas de instalações em laboratórios e hospitais de dimensão, organização e especialização diferentes, em 3 continentes, tornaram o Clinidata® um produto altamente inovador, inteligente, fiável e de máximo desempenho que a Maxdata se esforça por manter como referência superior na sua área.

A aposta na inovação é constante. Em 4 décadas, a Maxdata lançou quatro gerações do Clinidata®. A 4ª geração do Clinidata® integra e aumenta o leque de funcionalidades das aplicações antigas e dá um 'salto de gigante' na usabilidade e na flexibilidade: software 100% web completamente inovador e sem paralelo a nível internacional. Alguns destaques: é seguro *by design* respeitando o Regulamento Geral de Proteção de Dados, permite a utilização de sistemas operativos e bases de dados open source, inclui a migração automática de aplicações concorrentes e está preparado para ser utilizado a partir da cloud.

1.3 Planeamento do Projeto

O planeamento do projeto foi divido em três principais fases: a fase de planeamento do projeto; a fase de implementação e a fase de testes e documentação.

Dada a situação pandémica 'COVID-19' vivida durante o período de realização deste projeto, o planeamento que tinha sido inicialmente definido acabou por sofrer um desvio de dois meses, o que levou a que a data de conclusão fosse alterada para 30 de setembro de 2020. As fases e tarefas planeadas inicialmente mantiveram-se, no entanto, como os meses de março e abril apresentaram uma produtividade mais baixa devido a constrangimentos provocados por esta situação, as tarefas que incidiam sobre estes meses foram prolongadas.

O cronograma da Figura 1.1 apresenta o planeamento com as alterações feitas ao planeamento inicial. De seguida, é feito uma descrição das fases e atividades, referindo quais as alterações que foram necessárias.

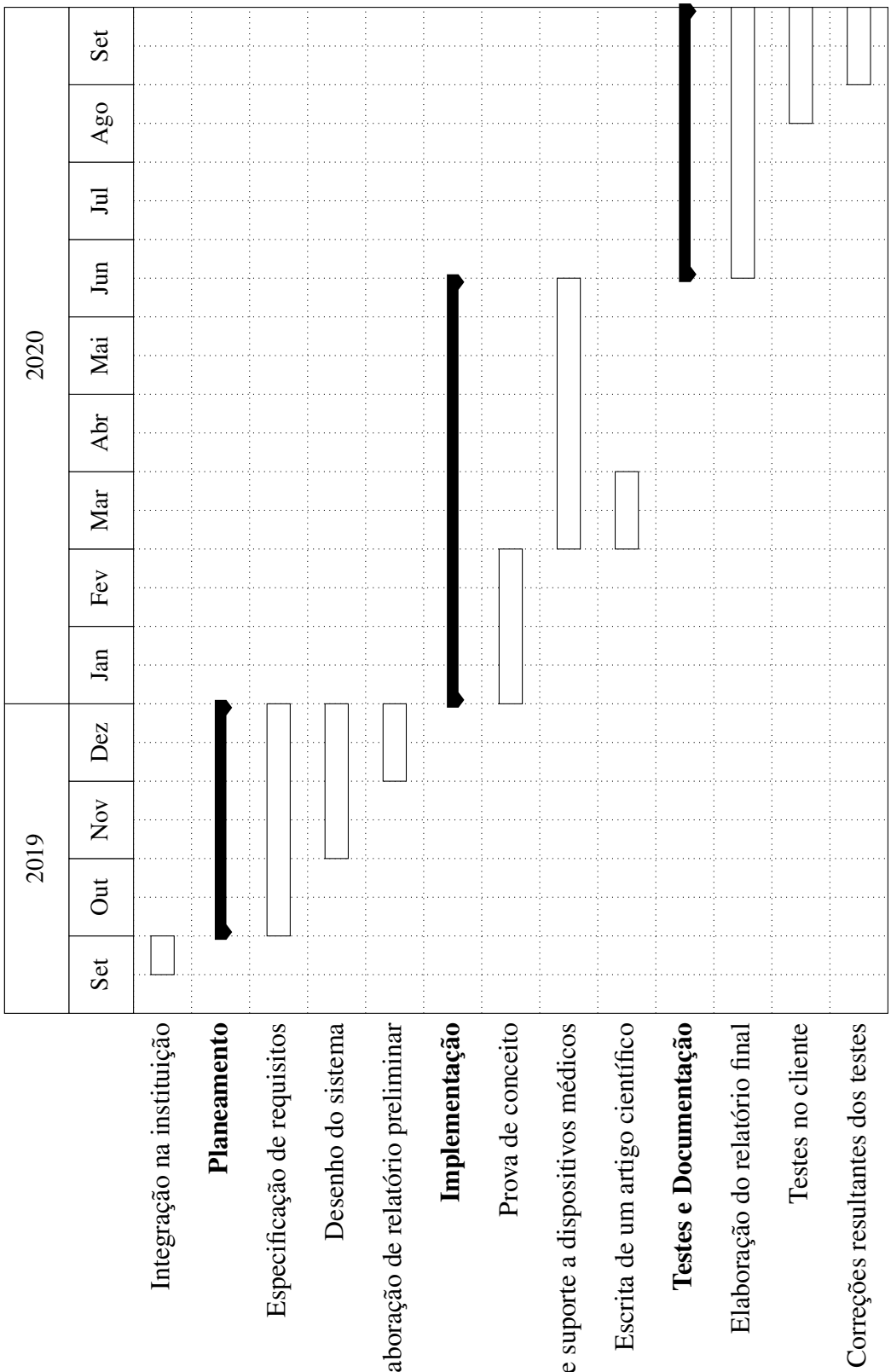


Figura 1.1: Mapa de Gantt do Planeamento do Trabalho

Previamente ao arranque do projeto, na segunda metade de Setembro, recebi orientação e formação por parte de vários colaboradores da Maxdata. Fiz uma leitura de toda a documentação sobre as tecnologias utilizadas, boas práticas e metodologias adotadas. Realizei ainda algumas tarefas da área de desenvolvimento do Clinidata®, nomeadamente tarefas de correção e adição de novas funcionalidades, que foram fundamentais para integração no ambiente da equipa de 'Investigação e Desenvolvimento' onde fui inserido.

Acompanhei um colega da equipa de 'Assistência e Implementação' ao Hospital de Vila Franca de Xira, onde tive a oportunidade de conhecer o trabalho realizado pelos colegas dessa equipa, a relação com o Cliente e o Processo Laboratorial de um Hospital, nomeadamente como funcionam os Dispositivos Médicos utilizados numa unidade laboratorial.

1º Fase (Setembro a Dezembro de 2019): Planeamento do Projeto

Esta fase cumpriu o planeamento original na integra, não havendo qualquer necessidade de alteração nas datas. A primeira tarefa desta fase consistiu na elaboração de uma lista de requisitos funcionais e não funcionais e teve a duração total desta fase, pois foi necessário uma constante revisão dos requisitos perante o desenho do sistema. Assim que se obteve uma primeira versão 'estável' de requisitos, iniciou-se a segunda tarefa, que consistiu no desenho do sistema. Paralelamente às restantes atividades, no último mês desta fase, foi elaborado um relatório preliminar deste projeto.

2ª Fase (Janeiro a Junho de 2020): Implementação

Esta fase iniciou-se em Janeiro e estava inicialmente prevista de ser terminada no final de maio, no entanto foi terminada no final da primeira metade de junho.

De forma a comprovar o funcionamento do sistema, nos meses de janeiro e fevereiro, elaborou-se uma 'prova de conceito' - implementação de um protótipo de alto nível - à qual se fez diferentes testes e se produziu um artigo científico durante o mês de março.

A tarefa de adição de suporte a dispositivos médicos estava inicialmente prevista de ser terminada no mês de abril. No entanto, devido às complicações anteriormente enunciadas, esta tarefa ficou terminada no final da primeira metade de mês de junho.

3ª Fase (Junho a Setembro de 2020): Testes e Documentação

O planeamento inicial definia que esta fase começava em maio e terminava em junho. Estas datas foram alteradas devido a terem sido adiados os testes no cliente como consequência da pandemia. Para além desta situação, o atraso nas fases anteriores levou a que esta fase tivesse inicio na segunda metade do mês de junho e conclusão no final de setembro.

Durante esta fase foi elaborado o presente relatório. Paralelamente, na segunda metade de agosto, deu-se inicio aos testes no cliente. As correções resultantes de problemas reportados dos testes no cliente constituíram a última tarefa a ser realizada.

1.4 Estrutura do Documento

Este documento está organizado da seguinte forma:

- **Capítulo 1:** Introdução ao projeto desenvolvido, os seus objetivos e estrutura do documento.
- **Capítulo 2:** Enquadramento a nível da área de negócio na qual o projeto se insere. Trabalho relacionado, onde é apresentado uma pesquisa de aplicações semelhantes à deste projeto e gestão do projeto em termos de metodologia e ferramentas utilizadas.
- **Capítulo 3:** Análise e Desenho do projeto onde são identificados os requisitos, diagramas, e outras análises feitas pré-implementação.
- **Capítulo 4:** Descreve o processo de implementação do projeto, detalhando cada funcionalidade e componente do projeto.
- **Capítulo 5:** Neste capítulo são apresentados as metodologias de teste utilizadas durante o desenvolvimento e os resultados de testes feitos no cliente.
- **Capítulo 6:** Considerações finais onde é feito um resumo do trabalho realizado e o trabalho futuro.

Capítulo 2

Sistemas de Informação Laboratorial

Neste capítulo descrevem-se as noções e conceitos base dos Sistemas de Informação Laboratorial. São enumerados e descritos os termos técnicos, tecnologias e sistemas que operam actualmente no área. O capítulo contempla uma revisão dos produtos semelhantes existentes no mercado e são discutidos os principais mecanismos necessários à implementação de novos produtos relacionados com este segmento de mercado.

Os principais produtos concorrentes ao Clinidata® são revistos, fazendo uma breve descrição das suas funcionalidades. A descrição é apresentada de forma sucinta uma vez que se baseia apenas na informação que cada fabricante disponibiliza na web, existindo pouca informação detalhada sobre os produtos. De seguida são discutidos os principais aspetos necessários ao desenvolvimento deste tipo de sistemas, apresentado de forma geral os conceitos envolvidos e mostrando vantagens e desvantagens do uso dos mesmos em produtos como o Clinidata®.

De forma a desenvolver o produto correspondente a este PEI, foram identificados produtos semelhantes através de uma pesquisa na *internet* e feita uma análise das funcionalidades consideradas como interessantes, com base na experiência da Maxdata e na análise de *software* concorrente, de forma a identificar pontos fortes e pontos fracos para fazer uma comparação com o módulo proposto. Foram analisadas três aplicações: Clinidata® XXI [16], Appolo [5] e Harvest [19]. O capítulo apresenta também a estrutura de gestão do projeto com destaque para as metodologias utilizadas na sua conceção e ainda uma síntese de tecnologias utilizadas.

2.1 Contextualização do Projeto

Este projeto surge de um trabalho autónomo realizado no âmbito do 'Projeto em Engenharia Informática (PEI)' da Faculdade de Ciências da Universidade de Lisboa. Foi realizado na instituição de acolhimento Maxdata Software, SA [1] e pode-se catalogar como uma solução de *Middleware* para a Área da Saúde, que tem como público alvo os Laboratórios de Análises Clínicas. Um dos principais objetivos deste Middleware é permitir a

comunicação de dispositivos médicos com o sistema de gestão laboratorial Clinidata®.

2.1.1 Laboratórios de Análises Clínicas

Numa visão simplificada, os laboratórios de análises clínicas, também designados como laboratórios de patologia clínica, podem ser vistos como produtores de resultados que são obtidos através da análise de amostras, normalmente fornecidas pelos pacientes que procuram os mesmos, com a finalidade de diagnosticar uma possível doença. Assim sendo, podemos considerar que estas instituições recebem como *input* os pedidos de exames e as respectivas amostras dos paciente (e.g. sangue, urina) e produzem como *output* relatórios desses mesmos exames com informação crucial para o paciente e médico responsável por realizar o diagnóstico. A elaboração destes relatórios é feita com base nos resultados obtidos em laboratório e num conjunto de informação sobre o paciente.

Atualmente, um pouco por todo o mundo, é comum existirem grandes grupos laboratoriais que possuem vários laboratórios distribuídos. É favorável para estes grupos que os mesmos possuam um *software* LIS centralizado de forma a armazenar informação, gerir e automatizar todo o processo por estes praticado, tornando transparente a distância entre os diferentes laboratórios.

Associado ao software LIS, existe o *hardware* fundamental para os laboratórios, os dispositivos médicos. Estes equipamentos produzem os resultados desejados através de uma amostra e da informação armazenada pelo LIS. Desta forma, podemos identificar que uma comunicação entre estes dois componentes é fundamental para tornar mais eficiente o processo laboratorial e resolver muitos outros problemas que surgem da não automação do mesmo.

O processo de análise uma amostra passa por um conjunto de etapas, formando um circuito. Este processo normalmente inicia-se com a prescrição de exames por parte de um médico que fará um diagnóstico a(os) paciente(s). Em exames em que são necessárias amostras do paciente é feita uma colheita num local que contém todos os recursos necessários, designado 'Posto de Colheita'. O responsável pela colheita utiliza o LIS para registar as amostras colhidas. Estas amostras são automaticamente associadas ao paciente e aos respectivos exames de requisição e recebem um código de identificação que é impresso, em formato código de barras, numa etiqueta e colado no tubo/recipiente que contém a mesma. A amostra segue então para a parte do laboratório onde se encontram os dispositivos médicos adequados para a realização dos exames requisitados, sendo esta a parte em que se foca este projeto. Após serem feitos todos os exames nos dispositivos médicos, estes devem ser exportados/inseridos no LIS para que possam estar acessíveis a todos os envolvidos responsáveis por produzir o relatório que por fim é dado a conhecer ao paciente.

2.1.2 Dispositivos Médicos

Os dispositivos médicos considerados neste projeto são dispositivos físicos que se encontram em laboratórios e que são capazes de produzir resultados com base numa amostra recebida.

De forma generalizada, o processo de utilização destes dispositivos inicia-se pela inserção de um tubo que contém a amostra. Esta amostra é identificada pelo dispositivo através do número de identificação da amostra, normalmente impresso numa etiqueta código-barras que é lido caso o dispositivo tenha um leitor adequado para tal. Caso contrário, pode ser inserido o número da amostra através de um teclado ou outra forma de *input* que o dispositivo disponibilize. Após a leitura deste número de identificação por parte do equipamento, segue-se a análise da amostra com base num conjunto de informações a que chamaremos de 'programação'. O momento em que é inserida a programação no dispositivo varia consoante as especificidades do mesmo e pode ser feito aquando da identificação da amostra ou previamente. A programação contém um conjunto de informações que leva o dispositivo a produzir resultados e está armazenada no LIS, sendo necessário a sua transferência para o dispositivo através de uma *interface* disponibilizada pelos mesmos.

Figura 2.1: Exemplo de Dispositivo Médico Analisador (ACL TOP)

Estes dispositivos juntam um conjunto de *hardware* especifico para realizar análises, fazer leituras, medir valores, entre outras especificidades. Além das funcionalidades especificas para realizar análises, existe ainda uma porta física de comunicação (e.g: RS232, Ethernet) para ligação a um terminal externo. Através da ligação a um terminal externo por esta porta, é possível que o dispositivo médico e o terminal consigam trocar mensagens entre si, através de um canal de comunicação que pode ser utilizado para o LIS

enviar programação para o dispositivo e o dispositivo produzir resultados e enviar para o LIS. Existem ainda dispositivos que possibilitam que sejam criados dois canais de comunicação: um exclusivo para a programação e outro exclusivo para os resultados.

O protocolo de comunicação utilizado por um dispositivo médico é normalmente descrito em manuais de serviço, que são em regra geral criados pelos fabricantes dos mesmos. Muitos destes dispositivos utilizam protocolos de comunicação proprietários, isto é, criados especificamente para aquele dispositivo ou para um conjunto de modelos da geração daquele dispositivo. No entanto alguns fabricantes optam por utilizar padrões (e.g: ASTM, HL7) que foram criados com a finalidade de acabar com a heterogeneidade existente e ainda existe alguns dispositivos que possibilitam que seja escolhido qual o padrão a utilizar na comunicação.

2.2 Clinidata® XXI

O Clinidata® XXI [16], antecessor do Clinidata®, é um sistema inteligente de gestão global de laboratórios de análises clínicas e de diagnóstico, nos aspectos técnicos e financeiros, desenvolvido pela instituição de acolhimento.

Este *software* já permitia a integração com dispositivos médicos auto-analisadores limitado às possibilidades da linguagem em que foi desenvolvido (Visual-Basic 6). O módulo de comunicação com os dispositivos ainda é utilizado nos dias decorrentes pela instituição de acolhimento e será substituído pelo módulo desenvolvido neste projeto.

Algumas das funcionalidades disponibilizadas por este produto relacionadas com este Projeto de Engenharia Informática são:

1. Comunicação com dispositivos.

2. Compatibilidade com Sistemas Operativos Windows.

3. Suporte de mais de 450 tipos de dispositivos diferentes.

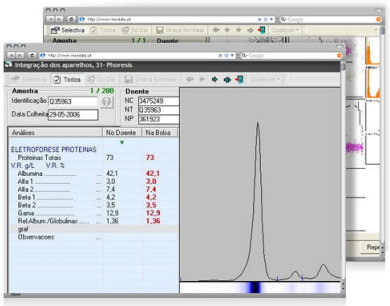


Figura 2.2: Interface do Clinidata® XXI

2.3 Appolo

O Appolo [5] é o principal produto comercializado pela empresa portuguesa Confidentia, encontrando-se instalado em mais de 120 laboratórios em Portugal, Angola e Moçambique. É um sistema de gestão global para o laboratório, reunindo todas as experiências e competências adquiridas pela equipa Confidentia.

Algumas das funcionalidades disponibilizadas por estes produto focado na gestão de dispositivos são:

1. Comunicação On-Line com qualquer dispositivo analisador que o permita.

2. Compatibilidade com Sistemas Operativos Windows.

3. Suporte de mais de 200 tipos de dispositivos diferentes.

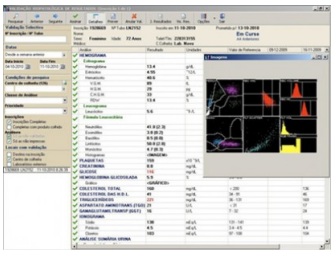


Figura 2.3: Interface do Appolo

2.4 Harvest

O Harvest [19], desenvolvido pela empresa norte-americana Orchard Software, é um sistema de gestão de laboratórios que maximiza processos automáticos e tecnologias de apoio à decisão para aumentar a eficiência do fluxo de trabalho de laboratório, redução de erros e aumento do valor de serviços de laboratórios a fornecedores e pacientes. Integrações de sistemas e *intefaces* de analisadores facilitam a automatização dos dados. O Harvest dispõe de ferramentas e tecnologias superiores para promover administração e resultados do contributo da organização para os cuidados de pacientes.

Este produto apresenta um módulo semelhante ao proposto por este projeto. Algumas das funcionalidades disponibilizadas por estes produto focadas no conceito deste Projeto de Engenharia Informática são:

1. Integração de Dispositivos Médicos com o LIS

2. Suporte de mais de 450 dispositivos

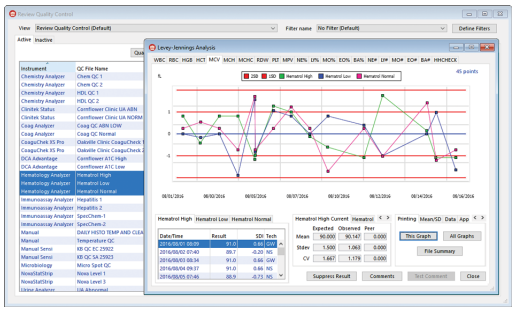


Figura 2.4: Interface do Harvest

2.5 Comparação de Funcionalidades

De forma a identificar as principais funcionalidades de um software clínico como os referidos anteriormente, foi efetuada uma comparação entre os diferentes softwares e identificados os principais componentes:

- **Comunicação com Dispositivos:** Capacidade de comunicar com dispositivo médicos para automatizar envio de programação e integração de resultados.
- **Atualizações Automáticas:** Existe a possibilidade de, através do produto, atualizar os protocolos dos dispositivos, sem a necessidade de parar o funcionamento de todo o sistema.
- **Cross-platform:** O produto e o respetivo módulo funcionam, pelo menos, nos sistemas Windows, Linux e MacOS.
- **Suporte a 450+ dispositivos:** Capacidade de suporte para mais de 450 modelos de dispositivos.
- **Modo Offline:** Todo o sistema mantém-se operacional mesmo em situações de falhas de comunicação com o produto, garantindo assim que os dispositivos continuam a escrever os resultados.

Com o estudo das soluções disponíveis no mercado, realizado nesta secção, foi elaborada a tabela 2.1 que indica quais os produtos que apresentam as funcionalidades idealizadas.

	Clinidata®	Clinidata® XXI	Appolo	Harvest
Comunicação com Dispositivos	✓	✓	✓	✓
Atualizações Automáticas	✓	✗	✗	✗
Cross-platform	✓	✗	✗	✓
Suporte a 450+ Dispostivos	✓	✓	✗	✓
Modo Offline	✓	✗	✗	✗

Tabela 2.1: Comparação de Funcionalidades

Com base nesta comparação podemos observar que o módulo proposto por este projeto coloca o Clinidata® à frente dos seus concorrentes apresentando um maior número de funcionalidades na comunicação com dispositivos, sendo duas destas funcionalidades inovadoras e não disponibilizadas por nenhum dos concorrentes considerados.

2.6 Gestão de Projeto

Nesta secção é apresentado o processo de gestão de software utilizado pela Maxdata, na qual este módulo, ao qual chamamos 'Clinidata® Instruments' está a ser implementado. Utilizando a abordagem de *Pressman*[15], é possível descrever o processo de gestão de um projeto através de Pessoas, Produto e Processo.

2.6.1 Pessoas

Os objectivos e responsabilidades de cada elemento de um projeto na Maxdata estão definidas no documento 'Manual de Funções e Responsabilidades'[14], que está disponível no SGI (Sistema de Gestão Integrado) e acessível a todos os colaboradores. No âmbito deste projeto consideramos as seguintes pessoas envolvidas no projeto e as suas funções:

- **Gestor de Projeto:** define o processo de gestão do projeto e monitoriza a avaliação de desempenho do mesmo.
- **Diretor de Investigação e Desenvolvimento:** responsável pela gestão do circuito de desenvolvimento do software. Faz análise funcional do sistema e avalia os elementos da equipa de Investigação e Desenvolvimento.
- **Analista Sénior:** elabora trabalhos de consultadoria a clientes e participa no levantamento de requisitos do *software*. Um analista sénior é uma pessoa com vários anos de experiência e enorme conhecimento na área, sendo o que conhece melhor as necessidade dos Utilizadores.

- ***Code Reviewer:*** faz a revisão do código desenvolvido e aponta eventuais defeitos, assegurando assim que este cumpra os padrões estabelecidos.
- **Programador/Analista:** faz parte da equipa de Investigação e Desenvolvimento e é responsável pela arquitetura e desenvolvimento de software e também por análises funcionais ao sistema.
- **Consultor Funcional:** é responsável pela instalação do *software* no cliente e pela resolução de eventuais problemas técnicos que possam ocorrer. Forma também a equipa de testes, efectuando testes aos módulos da aplicação após o seu desenvolvimento. Os erros reportados resultantes desses testes são corrigidos pela equipa de Investigação e Desenvolvimento.
- **Cliente:** através da sua colaboração, ajuda a definir os requisitos. Sugere ideias e fornece casos de uso para a aplicação. São exemplos de clientes da Maxdata, os hospitais e laboratórios clínicos onde o produto se encontra instalado.
- **Utilizador:** pessoa que trabalha com a aplicação diariamente. Neste módulo desenvolvido, os utilizadores que interagem diretamente são apenas os consultores funcionais, mas de forma indireta enfermeiros, técnicos laboratoriais e administrativos dependem do funcionamento deste módulo para efetuar o seu trabalho.

2.6.2 Produto

Como referido anteriormente, o produto desenvolvido - Clinidata® Instruments -trata-se de um módulo integrador de dispositivos médicos para o software Clinidata® que funciona como um middleware e faz a integração dos dispositivos médicos com o software Clinidata®. Sendo este software uma aplicação web com a capacidade de comunicação através da rede/internet por chamadas a procedimentos remotos, o objetivo será aproveitar esta capacidade para fazer a comunicação entre o 'Clinidata® Instruments' e o Clinidata®. Podemos ainda considerar como parte deste produto as alterações internas necessárias (e.g: procedimentos remotos, modelo de dados, entre outras) à aplicação Clinidata®.

O 'Clinidata® Instruments' serve de 'ponte' entre dois pontos: o Clinidata® e os Dispositivos Médicos através da criação de canais de comunicação para o mesmo efeito. As mensagens trocadas por este canal de comunicação são traduzidas em tempo real por um componente denominado 'Protocolo' (do Dispositivo). Este protocolo é responsável por especificar o comportamento para determinado dispositivo (ou conjunto de dispositivos) que o utilizem.

Sendo a área da saúde uma área critica em que perda de informação pode causar prejuízos inquantificáveis, este produto conta ainda com mecanismos de tolerância a falhas que podem acontecer dada a arquitetura distribuída desta solução, prevenindo assim que

quebras na ligação entre o 'Clinidata® Instruments' e o Clinidata® não impliquem a paragem de operação dos dispositivos médicos.

2.6.3 Processo

O modelo de processo de desenvolvimento adoptado para este projeto é o mesmo utilizado pela Maxdata, baseado em metodologias de desenvolvimento *Agile*. Com base neste modelo é feito um planeamento inicial da funcionalidade (ou conjunto de funcionalidades) a desenvolver. Segue-se então uma iteração composta pelas fases de levantamento de requisitos, análise e desenho, codificação, teste e documentação. Caso cheguemos a uma interacção onde o planeamento estabelecido foi atingido, terminamos esta iteração. A Figura 2.5 ilustra o modelo de desenvolvido *Agile*.

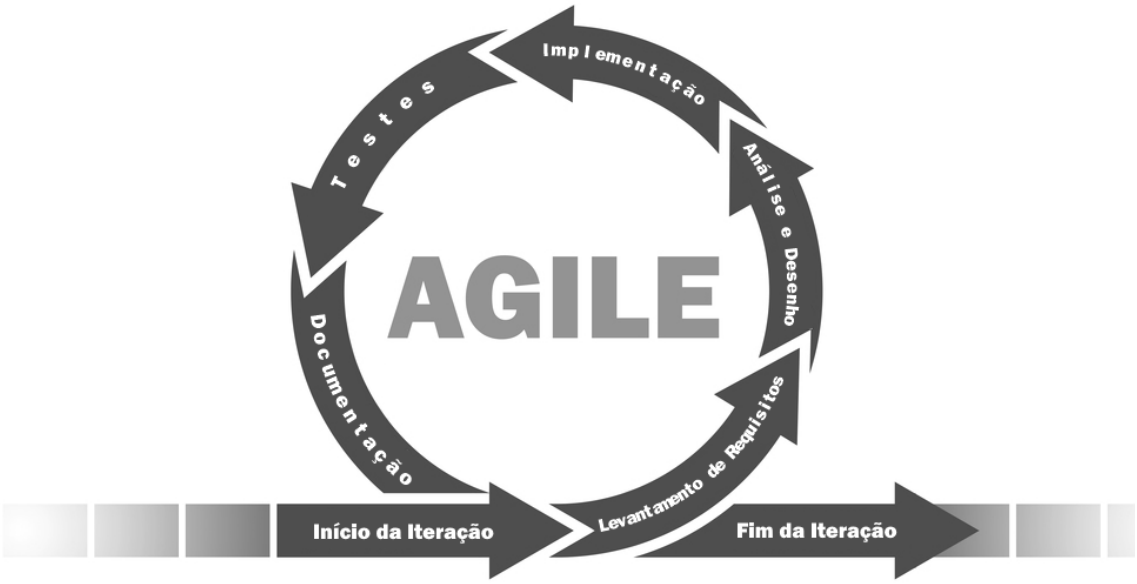


Figura 2.5: Modelo *Agile* adotado pela Maxdata e utilizado neste projeto

A fase de **Levantamento de Requisitos** é a primeira fase e tem como principal objetivo definir os requisitos das funcionalidades a desenvolver, assim como os seu detalhes, que muitas vezes são obtidos com base em reuniões com um ou mais clientes para assegurar os processos e regras de trabalho, assim como a legislação a aplicar.

A fase de **Análise e Desenho** define a forma mais correta de implementar os requisitos definidos anteriormente, assegurando que estes são compatíveis e coerentes com funcionalidades que já podem ter sido implementadas. Esta fase recebe como *input* a lista de requisitos estabelecidos anteriormente e devolve como *output* um documento onde são definidos o modelo de dados, diagramas UML e esboços de alto nível da interface. O documento referido servirá como base para a fase seguinte.

A fase da **Implementação**, ou Codificação, trata de implementar a funcionalidade ou corrigir eventuais erros detectados em testes feitos anteriormente. Assim sendo, podem existir dois tipos de *input:* o documento de Análise e Desenho e uma lista de erros detectados durante testes efectuados anteriormente. Na Maxdata, este processo é efetuado pela equipa de Investigação e Desenvolvimento e o código produzido nesta fase é colocado num sistema gestor de versões. Diariamente é enviado para toda a equipa de desenvolvimento um relatório com a contribuição de cada elemento, dando seguimento a uma reunião em que cada elemento partilha informações que considera relevantes sobre as suas contribuições. Nesta fase são ainda feitas revisões de código por parte dos *Code Reviewer* que analisam o código e devolvem *feedback* ao Programador.

A fase de **Testes** é feita por colaboradores, não responsáveis pela implementação da funcionalidade, que são os Consultores Funcionais que formam a equipa de testes. O objetivo desta fase é testar a funcionalidade implementada em diferentes ambientes próximos aos cenários de uma utilização real em ambiente de produção. No contexto do produto desenvolvido neste projeto, a testou testar os diferentes dispositivos médicos suportados, assim como as diferentes configurações dos mesmos consoante o ambiente laboratorial. Quando são reportados problemas nos testes, o Diretor de Investigação e Desenvolvimento revê os resultados e decide se a funcionalidade retorna para a fase de codificação.

A fase de **Documentação** poderá concluir a iteração e nesta fase é realizada uma atualização à documentação técnica e funcional da aplicação com base nos acontecimentos das etapas anteriores.

2.7 Tecnologias Utilizadas Durante o Projeto

Nesta secção é feita um resumo das tecnologias e ferramentas utilizadas na implementação do Clinidata®. É através destas ferramentas que são feitas as alterações na aplicação de forma a suportar o módulo desenvolvido.

2.7.1 Java

A aplicação foi desenvolvida com recurso a tecnologias Java a nível de *front-end* e de *back-end*. A utilização de Java em *front-end* é feita através do GWT [9], uma ferramenta que converte Java em JavaScript e HTML para que o programa seja executado sobre um browser. Desta forma é possível existir partilha de código entre cliente e servidor.

2.7.2 Hibernate

O Hibernate [10] é uma framework, escrita em linguagem Java, que faz o mapeamento entre os campos presentes nos diferentes objetos de negócio e as tabelas da base de dados.

Desta forma as diferentes classes Java da aplicação são representadas como tabelas de base de dados e criadas com um esquema que assegura equivalência.

Esta framework tem a sua própria linguagem para escrever *queries*, o HQL, que apresenta uma estrutura bastante semelhante ao SQL com a diferença que as queries são executadas sobre objetos mapeados pelo Hibernate. Outra vantagem do Hibernate é que fornece uma camada de abstracção entre a aplicação e o SGBD que permite que seja possível mudar o SGBD fazendo apenas alterações a nível de configuração.

Na área da saúde torna-se bastante vantajoso o uso do *Hibernate*, uma vez que este tipo de aplicações requerem constante adaptabilidade a novos paradigmas de desenvolvimento de software.

Outra vantagem é que esta framework facilita a conexão com a base de dados reduzindo o número de linhas de código necessárias para as operações CRUD (Create/Read/Update/Delete) aos dados e gere ainda as transações entre aplicação e base de dados de forma segura.

2.7.3 Liquibase

O Liquidbase [12] é uma solução simples, fiável e elegante para fazer o controlo de versão de Base de Dados através de ficheiros XML, YAML, JSON ou SQL e que suporta ainda diferentes versões de SGBD. Os ficheiros que contêm os ChangeSets são identificados por ID e Programador. Ao executar o ficheiro, as alterações à BD são efectuadas, sendo gerado um changelog na base de dados que identifica estas alterações. Na Maxdata utilizamos esta ferramenta para alterar a estrutura da Base de Dados da aplicação.

2.7.4 GWT

O Google Web Toolkit (GWT) é uma *framework* SOFEA (*Service-Oriented Front-End Architecture or Service Oriented User Interface*), *open-source* que permite o desenvolvimento de aplicações Web de grande escala com alto desempenho. Algumas das vantagens do GWT são:

- Facilita aos programadores o desenvolvimento da parte *front-end* da aplicação.
- Compila o código escrito em *Java* para *JavaScript*.
- Uma aplicação escrita em GWT é *cross-browser*, ou seja, compatível entre diferentes navegadores de *internet*. O GWT gera código *Javascript* adequado em cada navegador.
- É uma ferramenta open-source, completamente gratuita e já utilizada em enumeras aplicações.

Esta ferramenta contém um SDK que fornece APIs de Java e de Widgets, nomeadamente a MaxFramework, uma framework utilizada para componentes visuais da aplicação, abstraindo o programador as diferentes especificidades entre browsers e aumentando assim a sua produtividade de desenvolvimento.

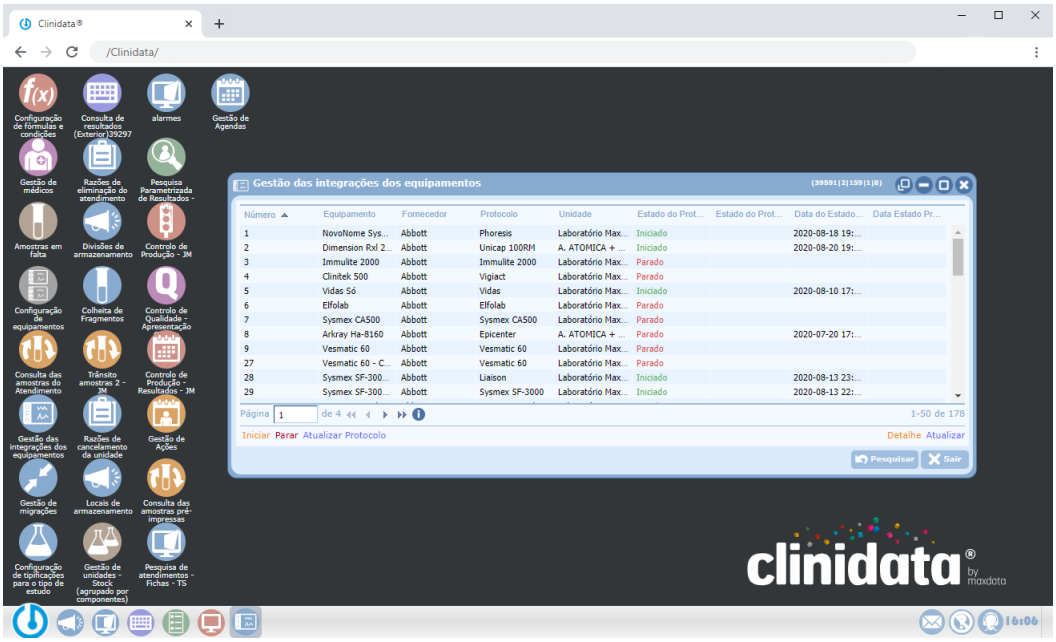


Figura 2.6: Interface gráfica do Clinidata® produzida pelo GWT

A Figura 2.6 mostra a *inteface* gráfica da aplicação web Clinidata®. Nesta figura encontra-se visível o ecrã de 'Gestão das integrações dos equipamentos' que foi produzido no contexto deste PEI para o módulo 'Clinidata® Instruments'.

2.7.5 *Spring*

O *Spring* [22] é uma framework open source para a plataforma Java que oferece um conjunto de funcionalidades, entre as quais destaca-se as anotações para injecção de dependências que se baseia no princípio de inversão de controlo, onde a classe não se configura a si própria, mas é configurada por outra classe. O *Spring* oferece gestão do ciclo de vida de objetos (designados por *beans*). Estes são configurados através de um ficheiro XML, onde são guardados as suas definições e dependências.

A anotação '@Autowired' fornece um contrutor 'injetável' automaticamente através da injeção de dependências. Quando se corre a aplicação Clinidata®, o *Spring* faz a correspondência entre as varáveis com a anotação '@Autowired' e os *beans* declarados no ficheiro. As variáveis são assim injetadas após a criação do *bean* e antes de algum método poder ser invocado. A anotação '@Service' identifica a classe Java como sendo um serviço.

Além da injeção de dependências, o suporte a transações é outra vantagem do *Spring*. Este oferece um gestor de transações que propaga uma transação a várias métodos e permite *roll-back* no caso de ter ocorrido uma exceção numa das ações do bloco. Esta *framework* permite criar uma cada de abstração que facilita o envio de dados provenientes do *fron-end* para o servidor com vista ao seu processamento. Esta camada de abstração é totalmente transparente para o cliente, não existindo quaisquer custos pela sua utilização.

2.7.6 Git/GitLab

O GIT [7] é um sistema gestor de versões distribuído adequado para grandes projetos, eficiente em recursos computacionais e com suporte para *branching* (suporte não-linear. Permite que diferentes programadores consigam trabalhar no mesmo projeto simultâneamente, garantindo a integridade dos ficheiros. Esta é a ferramenta utilizada na Maxdata para o projeto Clinidata®.

O GitLab [8] é um gestor de repositório de software baseado em GIT, com suporte a Wiki, gestão de tarefas e práticas de integração continua e entrega continua. O GIT é uma ferramenta de controlo de versões distribuída, usada principalmente no desenvolvimento de software. Garante a integridade dos ficheiros ao mesmo tempo que permite a programação simultânea por vários elementos da equipa de desenvolvimento. Outra vantagem é que, sendo um sistema distribuído e devido ao uso de repositórios remotos, existe sempre backups caso algo corra mal.

Este PEI levou ao surgimento de mais dois projetos no repositório GIT da Maxdata: o 'Clinidata Instruments Protocols' e o ' Clinidata Instruments Protocolos Manager'.

2.7.7 Apache Maven

O Apache Maven [3] é uma ferramenta que permite automatizar e gerir a compilação de qualquer projeto baseado em Java. Ao adicionar esta ferramenta a um projeto torna-se possível especificar a configuração de todo o processo de compilação do projeto, bem como as dependências necessárias ao mesmo, através de um ponto central de informação. Esse ponto de informação é um ficheiro com extensão XML (POM.xml) com a descrição da ordem de compilação de ficheiros, descrição das pastas necessárias para tal e dependências de módulos ou *plugins* externos. As dependências são automaticamente descarregadas e armazenadas no repositório local da máquina de desenvolvimento onde serão utilizadas.

Esta ferramenta é utilizada no projeto do Clinidata e nos projetos que foram criados para este Projeto em Engenharia Informática.

2.7.8 SQLite

O SQLite [20] é uma biblioteca escrita em linguagem C que implementa um sistema de base de dados SQL embutido. Os programas que utilizam esta biblioteca podem ter acesso a uma base de dados SQL sem executar um processo SGBD separado. Uma base de dados criada por esta biblioteca necessita apenas de um ficheiro em disco para armazenar os seus dados, sendo a biblioteca responsável pela leitura e escrita necessárias para realizar operações.

2.7.9 Apache Common Deamon

O Apache Common Deamon [2] é um conjunto de utilitários e classes de suporte *Java* para executar aplicações *Java* como processos de servidor.

Este suporte é feito em sistemas operativos *Windows* através do *Procrun*, que permite que as aplicações executem como serviços do *Windows*.

Para sistemas operativos *Unix*, o suporte é feito através de um conjunto de bibliotecas que formam o *Jsvc* e permitem que a aplicação *Java* corra como um *Deamon* do sistema operativo.

Capítulo 3

Análise e Desenho

A etapa de análise e desenho permitiu que fosse elaborado um planeamento do desenvolvimento da aplicação com base em casos de uso, que por sua vez levaram a uma especificação requisitos funcionais e não funcionais. Juntando a estas atividades uma análise ao *software* Clinidata® foi ainda possível definir a arquitetura e elaborar alguns diagramas que auxiliam a perceção do funcionamento do sistema e que servem como base para a fase de implementação.

3.1 Casos de Uso

Um diagrama de Casos de Uso é uma forma visual e primária de identificação de requisitos de um sistema que esteja em fase de desenvolvimento. Este diagrama representa o comportamento do software através da perspectiva do utilizador e da relação entre casos, actores e sistemas.

Na figura 3.1 é apresentado um diagrama de casos de uso para o sistema desenvolvido. São identificados neste diagrama dois atores, o Consultor Maxdata e o Dispositivo Médico que interagem com os seus casos de uso.

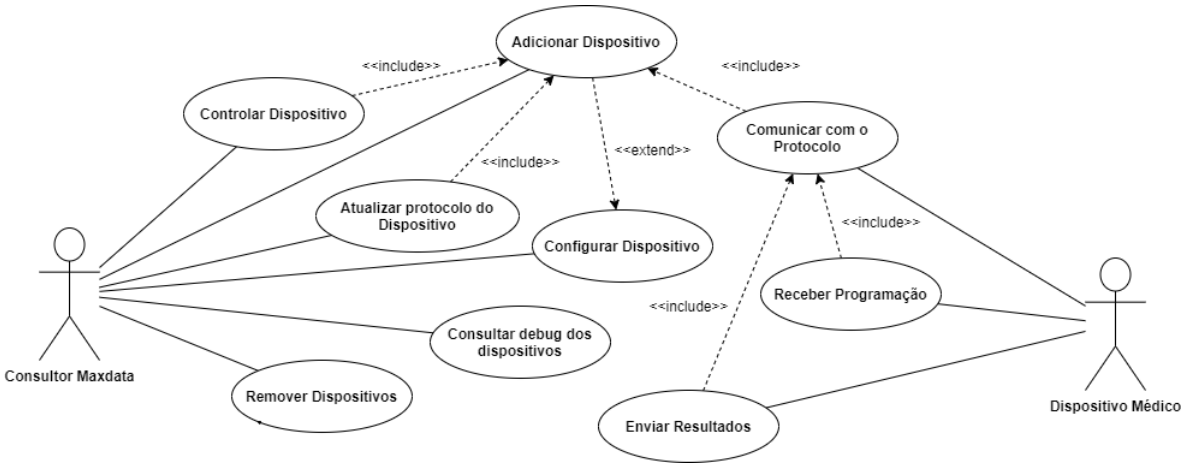


Figura 3.1: Diagrama de Casos de Uso do Sistema

Uma alternativa à representação dos casos de uso em formato de diagrama, é através de uma descrição narrativa da sequência de eventos que acontece quando um ator utiliza o sistema para realizar uma determinada tarefa. As seguintes tabelas apresentam esse tipo alternativo de representação.

Id:	UC1
Caso de Uso:	Adicionar Dispositivo
Objetivo:	Adicionar um dispositivo médico ao ao sistema
Ator:	Consultor Maxdata
Fluxo de Ações:	1. Aceder à aplicação Clinidata® 2. Entrar no ecrã de 'Configuração de Equipamentos' 3. Clicar no botão 'Inserir' 4. Preencher os campos de configuração do dispositivo e clicar em 'Guardar'
Resultado Previsto:	O novo dispositivo aparece selecionado na *grid* de dispositivos ativos e passa a ser detectado pelo 'Clinidata® Instruments'.
Estado de erro:	É apresentado uma mensagem que justifica o motivo pelo qual não foi possível adicionar o dispositivo.

Tabela 3.1: Caso de Uso: Adicionar Dispositivo

Id:	UC2
Caso de Uso:	Remover Dispositivo
Objetivo:	Remover um dispositivo médico do sistema
Ator:	Consultor Maxdata
Fluxo de Ações:	1. Aceder à aplicação Clinidata® 2. Entrar no ecrã de 'Configuração de Equipamentos' 3. Clicar no botão 'Eliminar'
Resultado Previsto:	O novo dispositivo desaparece na *grid* de dispositivos ativos, deixa de ser detectado pelo 'Clinidata® Instruments', sendo removido o registo do mesmo.
Estado de erro:	É apresentado uma mensagem que justifica o motivo pelo qual não foi possível remover o dispositivo.

Tabela 3.2: Caso de Uso: Remover Dispositivo

Id:	UC3
Caso de Uso:	Configurar Dispositivo
Objetivo:	Alterar a configuração de um dispositivo médico no Clinidata® e 'Clinidata® Instruments'

Tabela 3.3 – *Continua na página seguinte*

Tabela 3.3 – *Continuação da página anterior*

Ator:	Consultor Maxdata
Fluxo de Ações:	1. Aceder à aplicação Clinidata® 2. Entrar no ecrã de 'Configuração de Equipamentos' 3. Selecionar na *grid* o dispositivo a configurar e clicar em 'Alterar' 4. Fazer as respetivas alterações e clicar em 'Guardar'
Resultado Previsto:	A nova configuração do dispositivo é assumida pela aplicação.
Estado de erro:	É apresentado uma mensagem que justifica o motivo pelo qual não foi possível configurar o dispositivo. O 'Clinidata® Instruments' mantém a última configuração válida para o dispositivo.

Tabela 3.3: Caso de Uso: Configurar Dispositivo

Id:	UC4
Caso de Uso:	Consultar *Debug* do Dispositivo
Objetivo:	Consultar o *debug* produzido pelo protocolo do dispositivo
Ator:	Consultor Maxdata
Fluxo de Ações:	1. Aceder à pasta onde foi instalado o 'Clinidata® Instruments' 2. Abrir a pasta 'logs' 3. Abrir o ficheiro correspondente ao dispositivo 4. Identificar o problema
Resultado Previsto:	O consultor consegue identificar o problema no ficheiro de *Debug*.
Estado de erro:	O consultor não consegue identificar o problema no ficheiro de *Debug*.

Tabela 3.4: Caso de Uso: Consultar *Debug* do Dispositivo

Id:	UC5
Caso de Uso:	Controlar Dispositivo
Objetivo:	Arrancar ou parar o protocolo do dispositivo
Ator:	Consultor Maxdata

Tabela 3.5 – *Continua na página seguinte*

Tabela 3.5 – *Continuação da página anterior*

Fluxo de Ações:	1. Aceder à aplicação Clinidata® 2. Entrar no ecrã de 'Gestão de Equipamentos' 3. Selecionar na *grid* o dispositivo e clicar no respetivo botão (Iniciar/Parar) 4. Aguardar pela confirmação que a ação foi efectuada
Resultado Previsto:	O *Manager* do Clinidata® Instruments altera o estado do protocolo no 'Servidor de Protocolos'. O estado do dispositivo na *grid* é atualizado para o novo estado definido.
Estado de erro:	O protocolo imprime uma mensagem no *debug* sobre o motivo pelo qual não conseguiu atingir o novo estado. Na *grid* do ecrã de 'Gestão de Equipamentos' no Clinidata®, o estado do dispositivo aparece como 'Parado'.

Tabela 3.5: Caso de Uso: Controlar Dispositivo

Id:	UC6
Caso de Uso:	Atualizar Protocolo do Dispositivo
Objetivo:	Atualizar a versão do protocolo que está a ser utilizado pelo dispositivo
Ator:	Consultor Maxdata
Fluxo de Ações:	1. Aceder à aplicação Clinidata® 2. Entrar no ecrã de 'Gestão de Equipamentos' 3. Selecionar na *grid* o dispositivo e clicar em 'Atualizar Protocolo' 4. Aguardar que a confirmação de que o protocolo do dispositivo foi atualizado
Resultado Previsto:	O 'Clinidata® Instruments' atualiza a versão do protocolo do dispositivo e em seguida inicia o mesmo, registando no *debug* a ação da atualização. O estado do dispositivo na *grid* aparece como 'Iniciado'.
Estado de erro:	O 'Clinidata® Instruments' regista no *debug* o motivo da falha. O estado do dispositivo na *grid* aparece como 'Parado'.

Tabela 3.6: Caso de Uso: Atualizar Protocolo do Dispositivo

Id:	UC7
Caso de Uso:	Comunicar com o Protocolo
Objetivo:	Trocar mensagens com o protocolo
Ator:	Dispositivo Médico

Tabela 3.7 – *Continua na página seguinte*

Tabela 3.7 – *Continuação da página anterior*

Fluxo de Ações:	1. Enviar e interpretar dados
Resultado Previsto:	O dispositivo consegue enviar mensagens para o protocolo e interpretar a mensagens enviadas pelo protocolo.
Estado de erro:	Ocorre um erro no canal de comunicação. O erro é registado no ficheiro de *debug*.

Tabela 3.7: Caso de Uso: Comunicar com o Protocolo

Os casos de uso 'Enviar Resultados' e 'Receber Programação' são casos especificos do caso 'Comunicar com o Protocolo', sendo equivalentes o Fluxo de Ações, Resultado previsto e Estado de Erro.

3.2 Requisitos Funcionais

Os requisitos funcionais definem funções (conjunto de entradas, comportamento e saídas) da totalidade de um sistema e dos seus componentes. Desta forma, o conjunto de todos os requisitos é essencial para descrever o comportamento do sistema perante as suas funcionalidades. Para além da descrição, os requisitos são identificados com um tipo obrigatório (M) ou opcional (O) e a sua origem poderá ser do cliente (C) - através de pedidos ou necessidades do cliente - ou interna (I) - estratégia definida pela empresa.

ID	Descrição	Tipo	Origem
RF1	Os protocolos dos dispositivos devem poder ser instalados em máquinas localizadas perto dos dispositivos ou em máquinas localizadas em *datacenters* distantes.	M	I
RF2	Os protocolos dos dispositivos devem guardar os resultados recebidos dos dispositivos numa cache local persistente antes de os enviarem para o Clinidata®. Caso existam problemas/cortes temporários na ligação de rede entre protocolos e Clinidata®, os protocolos devem assegurar que continuam a receber resultados dos dispositivos, enviando-os para o Clinidata® logo que a ligação seja restabelecida.	M	I
RF3	Os protocolos dos dispositivos devem implementar uma cache de programação para conseguir continuar a programar dispositivos mesmo que ocorram quebras temporárias na ligação entre os protocolos e o Clinidata®.	O	I
RF4	Os protocolos dos dispositivos devem comunicar com o Clinidata® através de WebServices.	M	I

ID	Descrição	Tipo	Origem
RF5	Os protocolos dos dispositivos devem arrancar automaticamente como um serviço, garantindo-se que caso se reinicie o servidor onde estão instalados, os protocolos arrancam automaticamente (i.e, sem qualquer intervenção humana).	M	I
RF6	Mesmo que não exista ligação ao Clinidata® no (re)arranque dos protocolos, os protocolos devem arrancar e aguardar pela ligação ao Clinidata® para iniciarem o seu funcionamento.	M	I
RF7	No momento do arranque, caso não exista ligação ao Clinidata® mas já tenha existido anteriormente uma ligação com sucesso ao Clinidata®, o protocolo deve assumir as configurações anteriores e usar a cache local para operar.	O	I
RF8	O *debug* produzido pelos dispositivos deverá ser acessível a partir de ecrãs do Clinidata®.	O	I
RF9	Os dispositivos deverão poder ser controlados remotamente a partir de ecrã do Clinidata®: verificar se estão a executar e parar / iniciar dispositivos. Este controlo remoto deverá funcionar de forma similar aos processos automáticos do Clinidata®, ou seja, em vez de ser o Clinidata® a enviar comandos ao sistema de protocolos, deverá ser o sistema a questionar periodicamente o Clinidata® se existem novos comandos.	M	I
RF10	Os protocolos deverão ter a capacidade de atualização automática da versão a partir do Clinidata® (que por sua vez se liga ao Clinidata®CRM ou outro repositório) ou a partir de ficheiro colocado em pasta local.	M	I
RF11	Cada dispositivo poderá ter um ou dois canais de comunicação. A forma de funcionamento de cada canal será especificada no Clinidata®. Os parâmetros do 1º canal são os existentes hoje em dia no Clinidata®: tipo de ligação, velocidade, IP, porto, etc. Os parâmetros do 2º canal terão de ser adicionados ao modelo de dados uma vez que hoje em dia não existem os campos correspondentes.	M	I
RF12	O sistema a implementar não deverá depender de interface gráfica para permitir a operação em sistemas operativos sem ambiente gráfico.	M	I

ID	Descrição	Tipo	Origem
RF13	Para permitir interação com os dispositivos e protocolos na máquina em que estão instalados, deverá existir uma aplicação independente que permita executar um conjunto de comandos sobre os protocolos: listar dispositivos existentes, parar dispositivo, arrancar dispositivo e atualizar protocolo do dispositivo.	O	I
RF14	Os protocolos deverão usar um ficheiro de configuração local com a informação mínima necessária para a sua operação.	M	I
RF15	As *labels* e mensagens produzidas pelos protocolos deverão ser em inglês e não deverão ser configuráveis. Todas as mensagens produzidas pelos protocolos deverão ser escritas no ficheiro de *debug* e precedidas pela data e hora atual tal como acontece em outros programas da Maxdata. O ficheiro de *debug* deverá ser criado automaticamente após o arranque do protocolo caso ainda não exista. O nome do ficheiro deverá conter o nome do protocolo e o ID do dispositivo no Clinidata®.	M	I
RF16	No arranque do protocolo, deverá ser impressa mensagem no ficheiro de *debug* com a seguinte informação: data e hora, nome do protocolo, IP da máquina local e tipo de ligação (para cada canal).	M	I
RF17	Os protocolos deverão estar preparados para comunicar com os dispositivos usando diferentes tipos de ligação: TCP/IP cliente, TCP/IP servidor, RS232 e partilha de ficheiros. Estes tipos de ligação poderão aumentar no futuro. Independentemente do tipo de ligação, o processamento das mensagens recebidas e das mensagens a enviar deverá ser exatamente o mesmo, ou seja, deverá ser usado o mesmo pipeline de validação e tratamento nos vários tipos de ligação atuais e futuros.	M	C

ID	Descrição	Tipo	Origem
RF18	Quando o tipo de ligação é TCP/IP cliente, o protocolo estabelece uma ligação com o dispositivo usando o endereço IP e porto especificados na configuração do Clinidata®. Caso não consiga estabelecer a ligação, deverá tentar novamente em intervalos periódicos até conseguir estabelecer a ligação. Em cada tentativa, imprimir uma mensagem de erro no ficheiro de *debug* indicando o IP e porto para o qual não conseguiu estabelecer a ligação. Quando conseguir estabelecer ligação, imprimir uma mensagem de sucesso no ficheiro de *debug* indicando o IP e porto para o qual conseguiu estabelecer a ligação.	M	C
RF19	Quando o tipo de ligação é TCP/IP servidor, o protocolo faz *bind* a um Porto local em todas as interfaces de rede e aguarda ligação do dispositivo. Caso não consiga realizar *bind* ao Porto, imprimir mensagem de erro no ficheiro de *debug*. Caso consiga, imprimir uma mensagem de sucesso no ficheiro de *debug* indicando o IP e Porto em que foi realizado *bind*. Quando o dispositivo estabelecer uma ligação com sucesso, imprimir mensagem no *debug* com a informação de que a conexão foi estabelecida no respectivo IP e Porto.	M	C

ID	Descrição	Tipo	Origem
RF20	Para cada protocolo X desenvolvido, codificar um 2º protocolo espelho X-mirror que serve para testar o protocolo X. O protocolo espelho X-mirror deverá produzir as tramas principais do dispositivo para possibilitar a realização de testes básicos iniciais do protocolo X. A configuração/operação dos 2 protocolos depende do tipo de ligação: • Se o protocolo X for TCP-IP cliente, o X-mirror será TCP-IP servidor • Se o protocolo X for TCP-IP servidor, o X-mirror será TCP-IP cliente • Se o protocolo X for RS232, o protocolo X-mirror será também RS232. Neste caso usar software que gere portas série virtuais e criar uma ligação entre as 2 portas COM usadas pelos protocolos X e X-mirror. • Se o protocolo X for de partilha de ficheiros, o protocolo X-mirror será também de partilha de ficheiros, usando os mesmos caminhos/endereços de ficheiros de programação e receção do protocolo X.	O	I
RF21	Centralizar o código que é comum para o uso entre os diferentes protocolos e programar em cada protocolo apenas a componente específica desse protocolo.	M	C
RF22	Após a gravação de resultados via webservices, deve ser desencadeada a integração de resultados, para que os atuais processos automáticos de integração de resultados de dispositivos passem a ser necessários apenas em cenários muito específicos.	O	I

Tabela 3.8: Requisitos Funcionais

3.3 Requisitos Não Funcionais

Os requisitos não funcionais expressam qualidades globais ou atributos do sistema, definindo restrições no produto a ser desenvolvido e/ou no seu processo de desenvolvimento. Para além da descrição, os requisitos são identificados, à semelhança dos requisitos funcionais, com o tipo obrigatório (M) ou opcional (O) e a sua origem, que poderá ser do cliente (C) ou interna (I).

ID	Descrição	Tipo	Origem
RNF1	Os protocolos dos dispositivos devem ser codificados numa linguagem de programação *cross-platform* (que suporte, pelo menos, os sistemas operativos Windows, Linux e MacOS). Desejavelmente Java para reaproveitar o know-how existente na equipa de implementação sobre Java.	M	I
RNF2	O código dos protocolos dos dispositivos deve ser ofuscado para dificultar *reverse-engineering*.	O	I
RNF3	Os protocolos devem garantir uma resposta rápida aos dispositivos médicos, de forma a evitar *timeouts* provocados por respostas demoradas. A resposta deve ser dada na ordem do segundo.	M	I

Tabela 3.9: Requisitos Não Funcionais

3.4 Análise do Software Clinidata® LIS

O Clinidata® é uma aplicação web que já se encontra num estado de desenvolvimento bastante sólido e avançado. De forma a compreender quais as alterações necessárias a fazer para fazer cumprir os requisitos listados nas Seções 3.2 e 3.3, foi feita uma análise à atual arquitetura do Clinidata®.

3.4.1 Arquitetura do Clinidata®

A nível estrutural a aplicação encontra-se dividida em camadas que vão abstraindo operações sobre determinadas fontes de dados. No caso do Clinidata®, as fontes de dados podem ser bases de dados em Oracle, PostgreSQL ou qualquer outro tipo de base de dados que seja compatível com a tecnologia *Hibernate*.

A camada de dados disponibiliza um conjunto de *interfaces* que formam uma camada de abstração sobre as fontes de dados. É nesta camada que são criadas as *queries* HQL que funcionam com o *hibernate* e retornam os objectos Java equivalentes.

A camada de negócio disponibiliza as interfaces dos serviços que são constituídos por uma série de métodos expostos de modo a que possam ser invocados pela camada do cliente (através de RPC) e pela camada de integração por aplicações externas (através de SOAP).

A camada do cliente é responsável por criar a visualização que o cliente quando acede a partir do navegador web, sendo que a implementação desta camada é facilitada com o recurso ao GWT. Segue o modelo MVP (Model-View-Presenter), onde *model* representa o modelo de dados do domínio e os objetos e interfaces a serem exibidos. A *view* define

a interface gráfica onde são exibidos os dados do *model* e dispara eventos consoante os comandos/operações executadas pelo utilizador. O *presenter* obtém os dados do *model* e injeta na *view*. O *presenter* é o componente que dispõe da lógica correspondente ao domínio da aplicação. As interações do utilizador com a *view* desencadeiam ações a serem executadas no *presenter*.

A camada de integração é composta por um conjunto de Web Services que expõem um conjunto de métodos para aplicações externas acederem através do protocolo SOAP.

A Figura 3.2 representa a estrutura de camadas da aplicação.

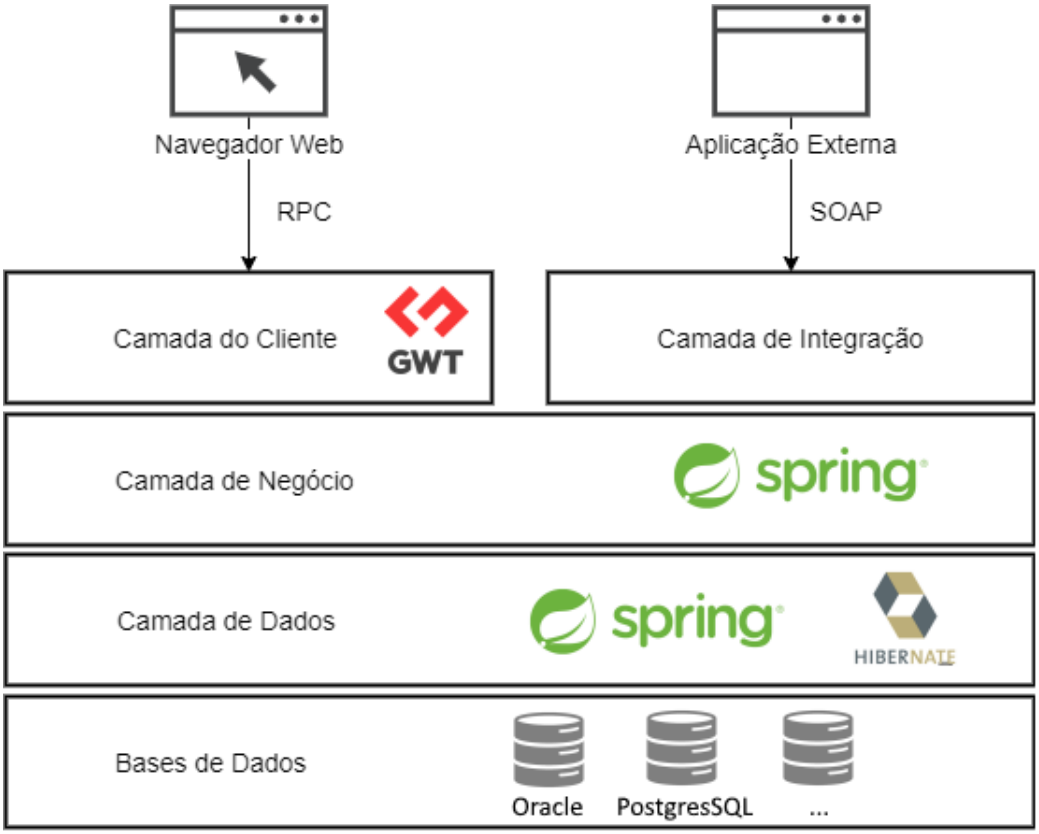


Figura 3.2: Arquitetura da Aplicação Clinidata® em Camadas

Como resultado deste PEI, o módulo 'Clinidata® Instruments', é composto por alterações nas camadas da aplicação enunciadas anteriormente e por uma aplicação externa que utiliza a camada de integração para comunicar e utilizar os recursos disponíveis no Clinidata®. Esta comunicação é possível através dos Web Services disponibilizados pela camada de integração através da troca de mensagens SOAP.

3.4.2 Modelo de Dados

Conteúdo oculto por motivos de confidencialidade.

3.5 Arquitetura do Clinidata® Instruments

Nesta secção é explicada a arquitetura do módulo Clinidata® Instruments e os respectivos componentes. As escolhas apresentadas foram maioritariamente com base nos requisitos mas também com base numa pesquisa e testes de performance a diferentes alternativas.

A primeira decisão tomada teve em consideração o RNF1 e levou a que o 'Java' fosse a linguagem de programação escolhida para as implementações necessárias, visto o java ser uma linguagem de programação poderosa e bastante popular. Esta é também a linguagem de programação utilizada no Clinidata® e nas aplicações mais atuais da Maxdata. Sendo a linguagem de programação dominada pelos restantes elementos da equipa de desenvolvimento da Maxdata.

Como referido anteriormente, o *software* desenvolvido tem de permitir a comunicação entre Dispositivos Médicos e o Clinidata®. Esta comunicação é feita por um componente que faz a ligação entre estes dois pontos, chamado de 'Protocolo'. A comunicação entre um Protocolo e o Clinidata® deve ser feita como especificado pelo RF4, ou seja, através de Web services, o que permite apenas que seja o Protocolo a fazer um pedido (e a receber a devida resposta) e nunca o Clinidata® a fazer um pedido ao Protocolo. Estes pedidos são feitos através do protocolo HTTP, o que permite que os protocolos possam ser instalados em qualquer localização desde que haja acesso pela rede (local ou internet) à aplicação Clinidata®.

Os dispositivos médicos encontram-se, normalmente, em laboratórios. A estrutura de um laboratório pode variar muito consoante a organização, mas existem terminais nestes laboratórios aos quais são ligados conjuntos de dispositivos médicos, como descrito anteriormente na secção 2.1.2. O protocolo tem ainda de estar preparado para comunicar com os dispositivos de três formas distintas: RS232, TCP/IP e Partilha de ficheiros, de acordo com o RF17. Deve ser assim instalado, em cada terminal, um protocolo correspondente por cada dispositivo ligado. Os protocolos devem ser executados como 'Serviços' do Sistema Operativo, satisfazendo o RF5. Estes terminais têm também acesso à internet, sendo que conseguimos também estabelecer ligação entre o Protocolo e o Clinidata®, respeitando o RF1 e RF4 simultaneamente. Cada protocolo deve ser ainda independente dos restantes para no caso de falha num protocolo, os restantes continuarem o seu normal funcionamento, pelo que definiu-se que cada protocolo deve ser um processo independente no sistema operativo.

De forma a cumprir com os requisitos RF9 e RF10 criou-se uma outra aplicação, que será um único processo adicional por cada máquina 'servidor de protocolos' ao que chamamos de 'Manager'. Esta aplicação gere qual o estado dos dispositivos para a máquina onde está instalado, consultando periodicamente o estado definido na aplicação Clinidata®. Esta aplicação executará também como 'Serviço' do Sistema Operativo e também tem a funcionalidade de atualizar a versão dos protocolos (quando definido para tal ação no Clinidata®), fazendo a transferência dos mesmos através de um repositório HTTP

definido.

Outro componente a adicionar a arquitetura é uma cache local, como especificado pelo RF2 e necessário também para o RF7. Para implementar esta cache foi utilizado o motor de base de dados SQLite. Uma das vantagens desta tecnologia é o facto de ser uma base de dados SQL de ficheiro, *self-contained* e que não requer servidor, assim sendo, podemos facilmente garantir uma base de dados embutida para cada protocolo, reforçando mais uma vez a independência de cada protocolo. A conecção entre a base de dados SQLite e o respetivo protocolo é assegurada pelo driver JDBC [21] do SQLite.

A figura 3.3 representa um diagrama da arquitetura do sistema com os componentes anteriormente referidos. Este diagrama permite visualizar como cada componente se liga entre si.

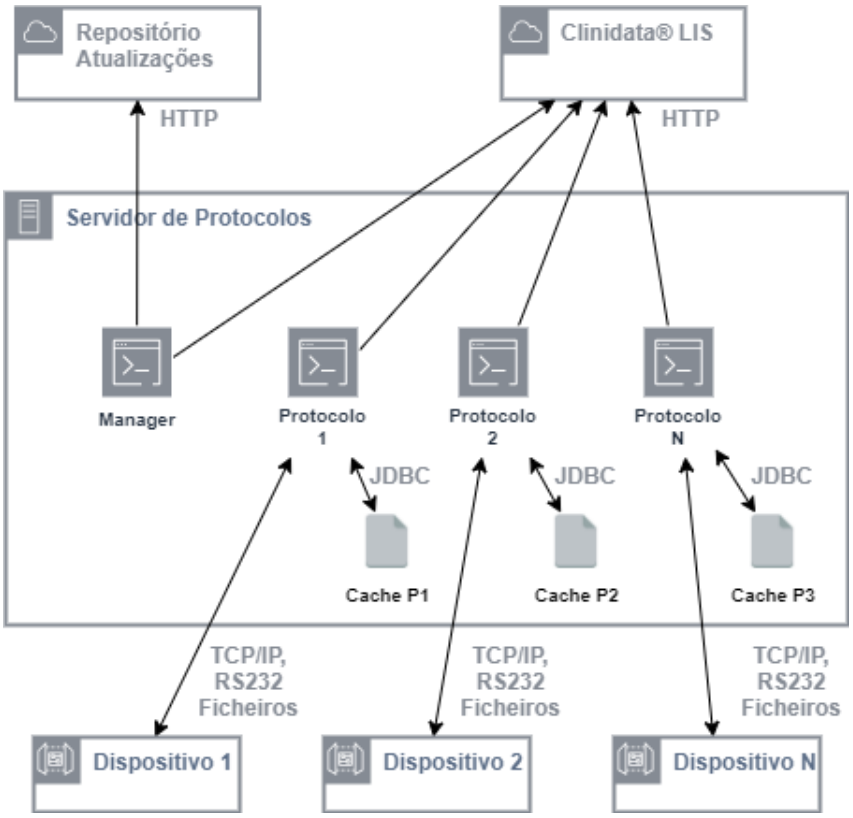


Figura 3.3: Diagrama da Arquitetura do Sistema

3.5.1 Diagramas de Classes

Conteúdo oculto por motivos de confidencialidade.

Capítulo 4

Implementação

A fase de implementação consistiu em converter a solução definida nas fases anteriores em *software*. Esta fase foi a mais extensa do projeto.

Para explicar as atividades desenvolvidas nesta fase, fez-se uma divisão deste capitulo em três secções: as modificações feitas no *software* Clinidata que permitem a comunicação do *middleware* desenvolvido; a implementação da estrutura genérica de um protocolo (Clinidata Instruments Protocols) e a implementação da aplicação desenvolvida para instalar em cada servidor de protocolos que é responsável pela gestão dos mesmos (Clinidata Instruments Protocols Manager).

4.1 Alterações no Clinidata®

A análise feita ao Software Clinidata, descrita na Secção 3.4, permitiu identificar quais as alterações necessárias para fazer o sistema funcionar de acordo com os requisitos especificados. Dado que a arquitetura definida para o sistema estabelece uma comunicação entre os componentes Clinidata com os componentes Protocolos e *Manager* através de Web-Services, a maioria das alterações foram na camada de integração e no modelo de dados. Foram ainda feitas alterações nas camadas intermédias (camada de negócio e camada de dados).

Conteúdo oculto por motivos de confidencialidade.

4.2 Clinidata® Instruments: Protocolos

O conjunto de aplicações que correm como processos independentes, também chamados de Protocolos e que foram desenvolvidas para este PEI fazem parte de um projeto *Java* chamado 'Clinidata Instruments Protocols'.

Conteúdo oculto por motivos de confidencialidade.

4.3 Clinidata® Instruments: Gestor de Protocolos

O 'Clinidata® Instruments Manager', ou apenas 'Manager' - como o nome indica - funciona como um gestor de protocolos. Esta aplicação deve ser instalada no servidor de protocolos para permitir que seja feito o controlo do estado dos protocolos dos dispositivos médicos que lhes estão associados, através do Clinidata.

Conteúdo oculto por motivos de confidencialidade.

Capítulo 5

Avaliação

O desenvolvimento de *software* está sujeito ao aparecimento de situações não planeadas que podem levar à necessidade de fazer alterações no produto que não foram expectáveis durante o planeamento. De modo a garantir a qualidade do produto desenvolvido e seguindo a metodologia *Agile* utilizada para a elaboração deste projeto, devem ser realizados testes ao longo do ciclo de vida do projecto. Para o projeto implementado foram realizados dois tipos de testes: os testes realizados em ambiente de desenvolvimento e os testes realizados no cliente.

5.1 Testes durante o Desenvolvimento

Os testes realizados em ambiente de desenvolvimento foram feitos na ausência de dispositivos médicos, que devido a um conjunto de factores externos, a Maxdata não possui. Isto obrigou a que fossem criados mecanismos de simulação dos mesmos de forma a permitir testar a sua interação com o sistema desenvolvido. Estes testes serviram também para detectar o máximo de erros possíveis antes de iniciar os testes no cliente.

5.1.1 Mecanismos de Simulação

Com o objetivo de testar os componentes do sistema durante a sua implementação recorreu-se a um conjunto de mecanismos que permitem simular a existência de um dispositivo a comunicar com o respectivo protocolo. Como especificado no requisito RF20, a criação de um 'Protocolo Espelho' faria com que fosse desenvolvido uma aplicação que se ligasse aos canais de comunicação do protocolo, enviando dados que provocariam alterações de estado nos protocolos. No entanto, sendo este um requisito opcional, decidiu-se recorrer ao uso do *software* 'Com Port Data Emulator' [18] para enviar dados através de um canal RS232 ou TCP/IP. Quanto à comunicação por partilha de ficheiros, para simular este tipo de comunicação, basta configurar uma pasta partilha e criar os ficheiros, simulando assim a criação de um ficheiro por parte de um dispositivo.

Na atualidade é raro os computadores estarem equipados com portas serial RS232. Para simular este tipo de ligação foi necessário recorrer ao *software* 'com0com [4]' que permite a criação de portas virtuais.

Por fim, de forma a completar as componentes necessárias para fazer todo o sistema funcionar, foi utilizada uma instalação, num servidor independente, da aplicação Clinidata® com a versão mais recente e com uma base de dados fictícia mas funcional.

A combinação destes mecanismos permitiu que um conjunto de testes fosse realizado fora do cliente. Estes testes serviram assim para validar o funcionamento geral do sistema, bem como algumas especificações dos protocolos, de forma a diminuir a probabilidade de erros durante os testes no cliente.

5.1.2 Testes ao Sistema

Com todos os componentes do sistema implementados, efectuou-se uma lista de testes de modo a verificar se o sistema funcionava como um todo da forma que foi desenhado. Este tipo de testes permitiu averiguar a existência de eventuais falhas ou incoerências e identificar potenciais melhorias a fazer.

Teste 1: Instalação do *Manager* no Servidor de Protocolos

A realização deste teste permitiu verificar as etapas necessárias para fazer com o que 'Clinidata® Instruments' fique operacional num 'Servidor de Protocolos'. Na realização deste teste foi utilizada uma máquina para onde foi copiado o 'Clinidata® Instruments'. Esta máquina possui acesso através da rede à instalação do Clinidata® utilizado para os testes e tem instalado os *softwares* necessários para simular a comunicação de dispositivos médicos. Após esta instalação é possível usufruir naquela máquina de todas as funcionalidades implementadas na aplicação no contexto deste PEI. Só após conseguir passar este teste é que se torna possível efectuar os restantes testes.

Algumas falhas identificadas na instalação foram:

- Falta de certificado SSL na *keystore* do *Java*. No caso de ser utilizado ligação segura ao Clinidata®, sem certificado não é possível a comunicação entre o Clinidata® Instruments e o Clinidata®.

- Erros na alteração dos parâmetros do *script* de instalação, para corresponder com a situação de instalação. Este erro levou a que fosse implementado o instalador automático embutido no *Manager*, abolindo este problema.

- Percepção do nome dos parâmetros do ficheiro de configuração do *Manager*. Os nomes foram alterados e foram adicionados comentários explicativos.

Para além de testar a existência de falhas nas etapas de instalação do 'Clinidata® Instruments', este teste serviu ainda para perceber como melhorar a instalação de forma a diminuir a probabilidade de erro por parte do técnico responsável pela mesma.

Teste 2: Inserir/Remover Dispositivos

No ecrã de configuração de equipamentos, no Clinidata®, é possível adicionar e remover dispositivos. Estes dispositivos são associados a uma máquina, neste caso, ao 'Servidor de Protocolos' utilizados para os testes. O *Manager* que está em execução nesta máquina tem de reconhecer quando são adicionados novos dispositivos e também quando são removidos. Para testar esta capacidade, fez-se um conjunto de adições e remoções de dispositivos no ecrã do Clinidata® e em seguida verificou-se se as mensagens produzidas pelo *Manager*, no seu ficheiro de *debug* correspondiam com as ações feitas no ecrã do Clinidata®.

Com este teste foi possível validar a informação retribuída pelo *Webmethod* do Clinidata® responsável por retornar todos os dispositivos configurados no ecrã para aquela máquina e o código do *Manager* responsável por invocar esse método e registar a adição ou remoção de dispositivos consoante a resposta obtida.

Teste 3: Iniciar/Parar/Alterar Dispositivos

Este teste foi realizado apenas após a confirmação de que o teste de 'Inserir/Remover Dispositivos' de que não haveria falhas nessas ações, visto que não faz sentido, por exemplo, ter o protocolo de um dispositivo que foi removido iniciado. Para este teste instalaram-se manualmente os protocolos que foram utilizados para a sua realização. A instalação automática de protocolos através de um repositório (atualização) foi considerada num outro teste em separado.

Após o teste anterior, manteve-se um conjunto de dispositivos configurados ao qual se aplicou diferentes estados através do ecrã de gestão de equipamentos. De seguida utilizou-se o *debug* do *Manager* para verificar se eram recebidos esses estados. No caso de ter sido definido o estado 'Iniciado' a determinado dispositivo, utilizou-se o Gestor de Tarefas do Sistema Operativo para verificar que o processo correspondente encontrava-se em execução.

Devido ao *design* do sistema, a alteração de dispositivos (configuração) só é atualizada no protocolo, cada vez que este inicia, ou seja, caso se tenha feito uma alteração na configuração do dispositivo enquanto o mesmo se encontrava em funcionamento, essa alteração só é assumida no próximo inicio do protocolo. Assim sendo, este teste permitiu testar também essa componente.

Teste 4: Atualizar Protocolos de Dispositivos

Para testar a atualização de protocolos criou-se um repositório com versões de protocolos. Removeu-se quaisquer protocolos instalados na máquina de testes e iniciou-se um conjunto de dispositivos no ecrã de gestão de equipamentos do Clinidata®. Em seguida verificou-se se os mesmos foram correctamente instalados na máquina, permitindo a sua inicialização. Testou-se também a atualização de protocolos em diferentes estados para assegurar que a lógica definida no desenho do sistema tinha sido correctamente implementada.

Deste teste surgiu ainda uma correção de um caso não considerado: possível falha na transferência do executável do protocolo. Ao acontecer esta falha é expectável que o *Manager* retorne o protocolo para o seu estado e versão anterior. A lógica implementada apagava em primeiro lugar o ficheiro executável do protocolo quando era solicitado uma atualização e em seguida transferia a nova versão, sendo impossível retornar à versão antiga caso ocorresse falha. Assim sendo, deve ser feita primeiro a transferência da nova versão e só quando terminar com sucesso deve ser apagar a versão antiga.

5.1.3 Testes aos Protocolos

Estes testes pretendem analisar se a leitura por parte do protocolo das tramas enviadas pelo dispositivo médico é a correta e se a criação de tramas de programação por parte do protocolo e o seu envio para os dispositivos cumpre a estrutura e regras esperadas pelo mesmo. Sendo o protocolo um processo independente, pretendeu-se testar o seu comportamento com base nos requisitos que são esperados.

Teste 1: Funcionamento dos protocolos sem ligação ao Clinidata®

Para além do funcionamento ótimo esperado quando todos os componentes do sistema estão operacionais ao mesmo tempo, existem requisitos que prevêem falhas na ligação ao Clinidata®.

Com base nos requisitos RF6 e RF7, o primeiro comportamento a testar foi a obtenção da configuração guardada em cache, se existente. Para tal, criou-se e iniciou-se um protocolo de dispositivo. Em seguida, cortou-se a ligação da máquina ao Clinidata® e reiniciou-se a mesma, de forma a verificar que o protocolo recuperava o estado e configuração anterior.

Outra falha a prever, com base no requisito RF2, é assegurar a recepção de resultados durante cortes na ligação e o seu envio assim que voltar a ser estabelecida a sua ligação. Para tal, iniciou-se um protocolo, cortou-se a ligação da máquina ao Clinidata®, e através dos mecanismos de simulação enviou-se tramas que foram construído resultados. Verificou-se na *cache* os resultados produzidos e em seguida estabeleceu-se a ligação ao

Clinidata®. Finalmente verificou-se se os resultados integrados no Clinidata® eram os mesmos que estavam guardados na *cache*.

Teste 2: Testes à integração de resultados e programação

De forma atingir o objetivo deste teste, a Maxdata disponibilizou um conjunto de ficheiros de *debug*, onde foram registadas trocas de tramas entre dispositivos médicos e aplicações utilizadas pela Maxdata no passado, semelhantes a esta. Existe ainda, no repositório interno [14] da empresa, um conjunto de 'manuais de serviço' de cada dispositivo. Para testar os resultados produzidos por um protocolo, enviou-se as tramas para o protocolo e, de seguida, analisou-se os resultados produzidos pelo mesmo. De forma a confirmar se os resultados produzidos eram os correctos, utilizou-se, quando disponível, as explicações e exemplos dos manuais de serviço. Para testar as tramas de programação produzidas pelo protocolo, criou-se cenários na aplicação Clinidata® utilizada para testes e analisou-se se a estrutura e conteúdo das tramas produzidas para avaliar eram equivalentes àquelas registadas pelas outras aplicações e se a estrutura das tramas seguia a estrutura apresentada pelo manual de serviço, quando disponível.

5.1.4 Testes de Capacidade e de Carga

Com o principal objetivo de identificar os limites da aplicação desenvolvida foram realizados testes de carga. Estes testes permitiram verificar o desempenho de um sistema e definir quais os requisitos necessários para o correto funcionamento do mesmo. Pretendeu-se ainda avaliar o desempenho da arquitetura implementada. Para tal, criou-se uma série de testes. Os resultados destes testes levaram à produção de um artigo com o título 'Fault-Tolerant Architecture for Real-Time Control of Distributed Medical Devices', onde são descritas as atividades resultantes de testes de capacidade e cargas, os respectivos resultados e conclusões. Este artigo foi anexado a este documento, estando disponível no Apêndice B.

Nesta secção destacam-se os testes que permitem fazer uma avaliação geral do sistema. Para avaliar se o sistema desempenha o seu papel sobre carga, destacam-se dois testes: o tempo de resposta dos protocolos aos dispositivos e a latência na integração de resultados. Para ambos os testes, instanciou-se um número de protocolos (#) que executam em paralelo e recebem uma mensagem que gera um resultado.

Para a máquina utilizada nos testes (servidor de protocolos), espera-se que 100 protocolos a correr em paralelo seja um valor adequado para as especificações da mesma. Ainda assim testou-se com excesso de carga (250 protocolos) para avaliar a influência do pior caso nos resultados. Numa situação real, um servidor de protocolos nunca terá mais que 100 protocolos associados. Os restante detalhes dos testes e métricas encontram-se descritas no artigo.

Teste 1: Tempo de resposta dos protocolos aos dispositivos

Como ojetivo para este teste pretendeu-se medir o tempo de resposta dos protocolos aos dispositivos, a partir do momento em que foi gerado o resultado e gravado na *cache*. Os dispositivos médicos mais exigentes requerem que seja dada uma resposta no máximo em 3 segundos. Assim sendo, para avaliar que o sistema consegue responder dentro dos tempos permitidos, nenhum valor máximo de resposta deste teste deve exceder 3 segundos.

#	Média	Desv. Padrão	Máximo
1	0.04	0.33	7
5	0.07	1.00	34
10	0.06	0.83	29
50	0.21	3.56	169
100	0.59	7.84	526
250	2.45	28.24	2 474

Tabela 5.1: Resposta (em ms) aos dispositivos, após guardar resultado na cache

Conforme esperado, quando o número de controladores aumenta, o tempo de resposta também aumenta. No entanto, os valores médio e máximo ainda estão dentro do tempo de resposta apropriado para a maioria dos dispositivos usados em laboratórios. No caso de 250 controladores os resultados mostram um aumento significativo no tempo resposta. Esta é uma consequência de sobrecarga no *hardware* utilizado para servidor de protocolos nestes testes. O número de controladores que podem ser executados em um servidor depende das especificações do servidor. Tendo em consideração as especificações de *hardtware* do servidor utilizado, deve-se evitar a configuração de 250 controladores num servidor com as mesmas especificações. No entanto, mesmo em condições de sobrecarga, o tempo de resposta máximo foi inferior a 3 segundos, o que está em conformidade com o intervalo de 3-20 segundos típico da maioria dos dispositivos.

Teste 2: Latência na integração de resultados

Os resultados acumulados na *cache* são enviados para o Clinidata® a cada segundo. Este teste avalia qual o tempo que passa desde que o resultado foi produzido pelo dispositivo até estar disponível no Clinidata®.

#	Média	Desv. Padrão	Mínimo	Máximo
1	1 009.07	575.47	11	2 004
5	999.60	578.48	5	2 006
10	1 006.65	576.71	4	2 005
50	1 003.14	577.09	2	2 041
100	1 002.31	577.33	2	2 154
250	1 063.33	642.50	4	4 553

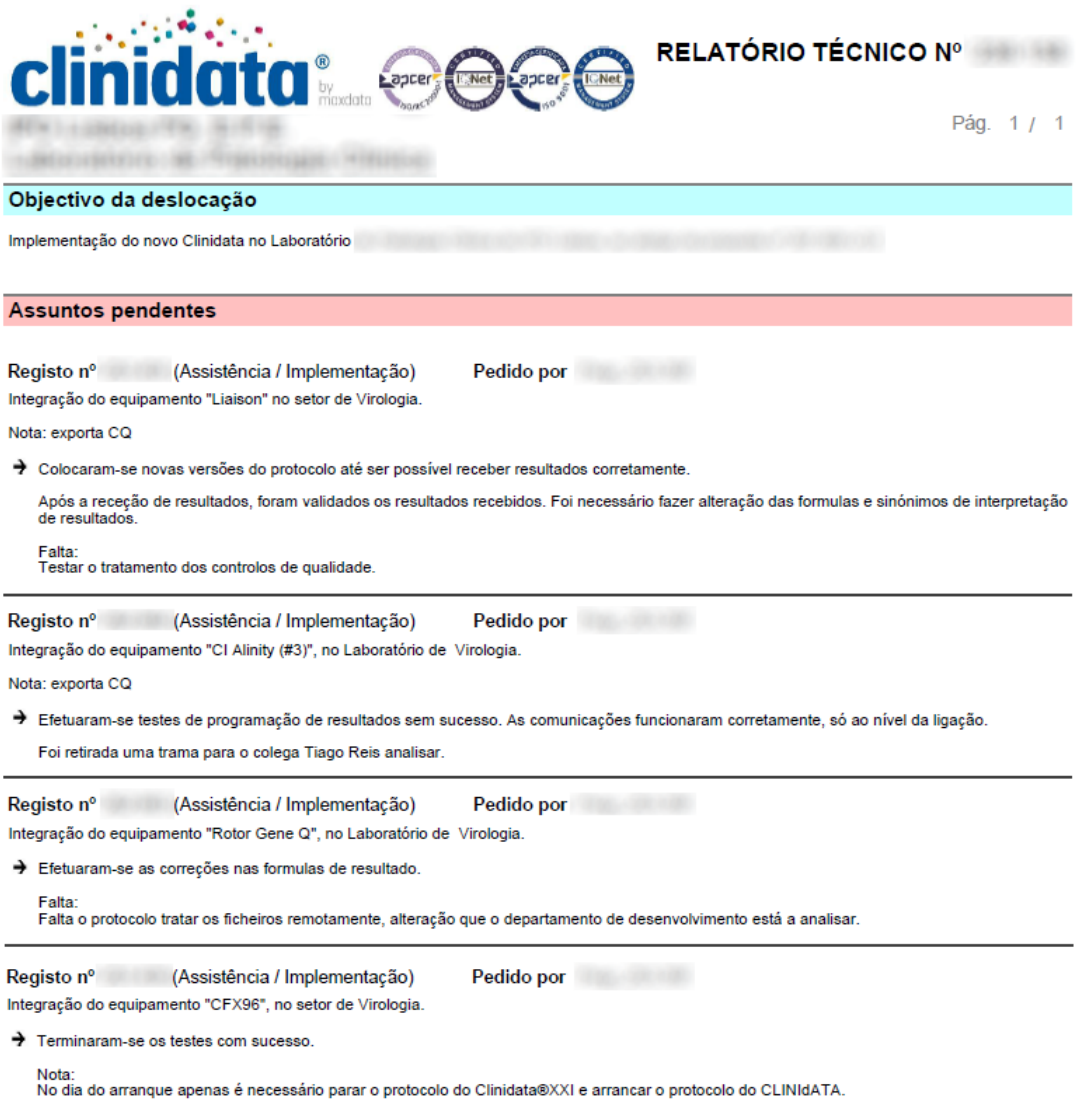
Tabela 5.2: Latência (em ms) na integração de resultados

Através da Tabela 5.2 podemos concluir que os resultados estão disponíveis no Clinidata®, num tempo médio de 1 segundo. Na mesma tabela também podemos ver que o máximo pode chegar a 4,5 segundos para 250 controladores. Esse tempo é alto e ocorre quando os resultados são armazenados no cache e o número de resultados na *cache* é maior que o número de resultados enviados em cada tarefa de envio. Nesta situação, os resultados são enviados apenas na próxima iteração da tarefa de envio que é usada para transmitir dados do servidor do controlador para o Clinidata®. Em uma situação real, o número de controladores é distribuído por servidores consoante as especificações dos mesmo, para evitar que aconteçam estas situações de excesso de carga.

5.2 Testes no Cliente

Os testes no cliente são realizados pela equipa de testes com recurso aos dispositivos médicos e possibilitam testar cenários que os testes em ambiente de desenvolvimento não permitem. Estes testes permitem uma aproximação do ambiente de produção na qual este produto será utilizado, sendo que o principal objetivo é testar principalmente a comunicação entre o protocolo e o dispositivo médico, para assegurar que a interpretação de mensagens recebidas pelo protocolo e as mensagens enviadas pelo protocolo ao dispositivo estão corretas.

Para realizar estes testes, a equipa desloca-se ao laboratório do cliente com um plano de testes para um ou mais dispositivos. Para testar um dispositivo conforme o planeado, é necessário que o cliente dispense esse equipamento durante o período de tempo necessário. Devido a estas condições, estes testes tornam-se mais demorados, pois nem sempre se consegue testar imediatamente uma nova funcionalidade ou correção. Até à data de escrita deste documento foram testados quase todos os equipamentos que se encontram na lista de protocolos desenvolvidos do Apêndice A.



clinidata® by maxdata

RELATÓRIO TÉCNICO Nº

Pág. 1 / 1

Objectivo da deslocação

Implementação do novo Clinidata no Laboratório

Assuntos pendentes

Registo nº (Assistência / Implementação) **Pedido por**

Integração do equipamento "Liaison" no setor de Virologia.

Nota: exporta CQ

→ Colocaram-se novas versões do protocolo até ser possível receber resultados corretamente.

Após a receção de resultados, foram validados os resultados recebidos. Foi necessário fazer alteração das formulas e sinónimos de interpretação de resultados.

Falta:
Testar o tratamento dos controlos de qualidade.

Registo nº (Assistência / Implementação) **Pedido por**

Integração do equipamento "CI Alinity (#3)", no Laboratório de Virologia.

Nota: exporta CQ

→ Efetuaram-se testes de programação de resultados sem sucesso. As comunicações funcionaram corretamente, só ao nível da ligação.

Foi retirada uma trama para o colega Tiago Reis analisar.

Registo nº (Assistência / Implementação) **Pedido por**

Integração do equipamento "Rotor Gene Q", no Laboratório de Virologia.

→ Efetuaram-se as correções nas formulas de resultado.

Falta:
Falta o protocolo tratar os ficheiros remotamente, alteração que o departamento de desenvolvimento está a analisar.

Registo nº (Assistência / Implementação) **Pedido por**

Integração do equipamento "CFX96", no setor de Virologia.

→ Terminaram-se os testes com sucesso.

Nota:
No dia do arranque apenas é necessário parar o protocolo do Clinidata®XXI e arrancar o protocolo do CLINIdATA.

Figura 5.1: Exemplo de um relatório de testes no cliente

Após cada deslocação da equipa de testes ao cliente, é produzido um relatório para cada dispositivo testado em que são enumerados os testes realizados e o seu resultado. A Figura 5.1 mostra um exemplo de um relatório produzido onde foram testados os protocolos 'Liaison', 'Alinity CI', 'Rotor Gene Q' e CFX96'. Os testes que falharam implicam que sejam feitas correções nos protocolo. Nesse caso é ainda anexado o ficheiro de *debug* que foi produzido pelo protocolo que, juntamente com o resultado esperado, permite que seja identificado o problema de forma a proceder à sua correção.

Até à data, foram testados 26 dos 37 protocolos implementados recorrendo a dispositivos dos dois laboratórios onde o Clinidata® Instruments se encontra instalado. A lista de protocolos testados no cliente encontra-se disponivel no Apêndice A.

Capítulo 6

Considerações Finais

No presente documento, foram apresentadas as atividades que possibilitaram o desenvolvimento do módulo 'Clinidata® Instruments', uma solução que permite a comunicação e controlo de dispositivos médicos analisadores em laboratórios. As decisões e escolhas de tecnologia e metodologias aqui apresentadas resultaram de um trabalho de pesquisa combinado com a experiência acumulada ao longo dos anos pela Maxdata nesta área.

Durante a realização do projeto surgiram algumas dificuldades que foram ultrapassadas com a ajuda e colaboração dos meus colegas de equipa. Estas dificuldades resultaram principalmente da minha falta de conhecimento na área de negócio e também na dificuldade de analisar e compreender o funcionamento dos diferentes dispositivos médicos para conseguir definir qual a lógica que deveria ser comum a todos os protocolos (código geral) e qual a que deveria ser específica.

O módulo 'Clinidata® Instruments' foi desenvolvido desde o planeamento até aos testes, encontrando-se neste momento a ser utilizado em ambiente de produção em dois laboratórios. Em termos de requisitos, todos os requisitos funcionais obrigatórios e todos os requisitos não funcionais foram implementados. Num total de cinco requisitos funcionais opcionais, foram implementados dois deles: RF7 e R20. Os requisitos funcionais opcionais RF8 e RF13 são de melhorias na interação com o utilizador e foram deixados para trabalho futuro. Os restantes requisitos - RF3 e R22 - não foram implementados porque, no caso do primeiro, seria preciso prever a que momento um dispositivo pede programação para determinada amostra, e no caso do segundo, porque se determinou que não se aplicava esta necessidade porque os processos automáticos de integração de resultados do Clinidata® continuam a funcionar através de *pooling*.

Em suma neste projeto foi realizado: alterações na aplicação Clinidata®, a nível de *front-end* e *back-end*, nas diferentes camadas da aplicação; o código geral que permite a integração com o Clinidata® - abertura de canais de comunicação com o dispositivo, processamento de resultados, envio de programação, entre outros - que contém a estrutura a implementar por cada implementação de protocolo; a implementação de 37 diferentes protocolos e a aplicação 'Manager' que permite que seja feito o controlo dos protocolos

através do Clinidata®. Juntando a estes componentes todo o planeamento, desenho e testes do sistema, considero que este projeto foi bem sucedido e os seus objetivos foram cumpridos.

6.1 Trabalho Futuro

Em termos de novas funcionalidades, será trabalho futuro a implementação dos restantes requisitos funcionais opcionais não implementados. Considero que o RF8 é o mais importante porque adiciona uma funcionalidade que é útil: conseguir consultar nos ecrãs do Clinidata® o *debug* produzidos pelos dispositivos pode ajudar a identificar um problema sem haver a necessidade de ser preciso aceder ao 'Servidor de Protocolos'. O requisito RF13 é útil em situações que haja cortes na ligação entre o 'Servidor de Protocolo' e o Clinidata® e o técnico precise de alterar o estado dos protocolos.

A criação de novos protocolos para suportar o leque de dispositivos pretendidos pela Maxdata é também um trabalho ainda por realizar e que é continuo devido às necessidades dos clientes se alterarem consoante o surgimento de novos dispositivos. É comum ainda ser necessário alterações aos protocolos já existentes para adicionar novas funcionalidades.

Por fim, tal como todos os *softwares* que acabam de ser desenvolvidos, existe todo um trabalho de manutenção a ser feito. Essa manutenção pode consistir em alterações no *software* para corrigir defeitos e deficiências que foram encontradas durante a utilização no cliente ou de adição de novas funcionalidades para melhorar o seu funcionamento ou usabilidade.

Apêndice A

Lista de Protocolos Desenvolvidos

#	Nome do Protocolo	Testado no Cliente
1	A Analyst	✗
2	Access	✗
3	ACL Top	✓
4	Alinity CI	✓
5	AVL Omni	✗
6	Baby	✗
7	Bactec 9000	✓
8	Balanca VWR	✗
9	BN ProSpec	✓
10	Cell-Dyn Sapphire	✓
11	Centaur CP	✓
12	CFX96	✓
13	CFX Connect	✓
14	Cobas E411	✓
15	D-10	✓
16	Diana 5	✗
17	Epicenter	✓
18	Evolis	✓
19	HS 8	✗
20	Immulite 2000	✓

#	Nome do Protocolo	Testado no Cliente
21	ImmunoCAP 250	✓
22	Indexor	✓
23	Liaison	✓
24	Menasoft	✓
25	MGIT 960	✗
26	Monitor-S	✗
27	Phoresis	✓
28	Radiance	✓
29	RapidCom	✓
30	Rotor Gene Q	✓
31	Seegene	✓
32	Sta Compact Max	✗
33	Unicap 100RM	✓
34	UrinQuick	✓
35	Vesmatic 60	✓
36	Vigiact	✓
37	Walkaway	✓

Tabela A.1: Protocolos Desenvolvidos

Apêndice B

Artigo ’Fault-Tolerant Architecture for Real-TimeControl of Distributed Medical Devices’

Fault-Tolerant Architecture for Real-Time Control of Distributed Medical Devices

Tiago Reis[1,2], Alexandre Correia[1], Paulo Sousa[1], and José Cecílio[2]

[1] Maxdata Software, SA - Carregado, Portugal
[2] LASIGE, Faculty of Sciences, University of Lisbon, Portugal

Abstract. Clinical laboratories use a myriad of medical devices (e.g., automated analyzers) to determine the results of its tests. Modern laboratories have evolved towards almost complete automation, by making use of laboratory information systems (LIS) that interact with medical devices in an automatic way using real-time communication protocols. However, laboratories have been increasing in terms of size and complexity mostly due to economic reasons: large groups of geographically distributed laboratories have been replacing small laboratories. This trend poses challenges for traditional LIS architectures based on a centralized system, given that the distributed nature of modern laboratories has a negative impact on the performance of the communication between medical devices and LIS, and also increases the risk of temporary network disconnection. In this paper we address these challenges by proposing a novel architecture that seamlessly integrates distributed medical devices using mechanisms that provide real-time operation and ensure fault-tolerance. The proposed system is expected to continuously provide trustworthy services even in critical scenarios.

Keywords: Medical Devices · Laboratories · Real Time · Distributed System · Fault-tolerance.

1 Introduction

The applications and deployment of ubiquitous systems are modifying the physical world, mainly medical environment. Smart sensing units with computational and decision-making abilities are integrated with the physical world entities to create new services [5]. These new services aim to offer benefits for end-users, commercial and residential consumers, scaling healthcare systems [4], [2]. The creation of geographically distributed laboratories groups is becoming a common practice [8], [7]. This new approach is replacing the traditional one where independent laboratories operate without sharing its information and/or resources. Currently, the laboratory processes are changing to add the possibility to offer more complex tests and to share knowledge along a group. One example is related with medical analyzer devices that are frequently used and its information must be shared by different laboratories to create a unified patient record [9]. In this context, geographically distributed laboratories groups are being formed, creating distributed laboratory information systems (D-LIS). However, new challenges

are emerging in these D-LIS. In order to deliver a certain degree of reliability, these new D-LIS must be able to deal with network, fault-tolerance and real-time issues.

There are multiple medical devices used to perform different types of tests in many specialties (e.g., biochemistry, hematology, microbiology). The diversity of devices available varies in terms of brand, model, communication technology and price. Typically, those devices are equipped with a physical port which allows to connect it to a host terminal for data transmission. Within the traditional approach of a single independent laboratory, that host terminal is typically in the same local network of the server where the LIS software is running, avoiding the need of other infrastructure for data communication or processing. However, this setup becomes inadequate for new D-LIS, where data needs to be transmitted over the internet. At the same time, laboratories expect to use medical devices as they were used before, carrying out its control through two basic operations: result and programming request are sent from the device to the D-LIS and configuration commands are sent from D-LIS to the medical device.

In this practical experience report we evaluate a fault-tolerant architecture for D-LIS, taking into account the importance of data availability on LIS server, the latency of network operation and the real-time requirements of laboratory devices.

2 System Architecture

Typically, in a D-LIS, devices are connected to the information system through a VPN over the internet. Many common medical devices that are used by laboratories do not have direct capabilities to establish a network connection, but all of them have a physical port (e.g., RS-232) that allows to connect to a nearby host or to a specific hardware (e.g., serial port server), which in turn expose the device to the network. There is a communication protocol that must be followed to read or send messages from or to the device. This protocol may follow a standard (e.g., ASTM [3], HL7 [6]) or it can be proprietary. There is not an universal solution.

In order to deal with these characteristics and to build a D-LIS, we need to design specific controllers to operate near each device. Each controller will be managed by a controller server which implements a network interface with D-LIS. The idea is that each local laboratory / local site must have at least one controller server which supports one or several controllers. It is intended that each controller operates independently. In the case of unpredictable problems in a controller (e.g., malfunction of the device), other controllers must be available to work without problems. To ensure this, each controller is deployed as a separate process that does not share any resource with the remaining controllers.

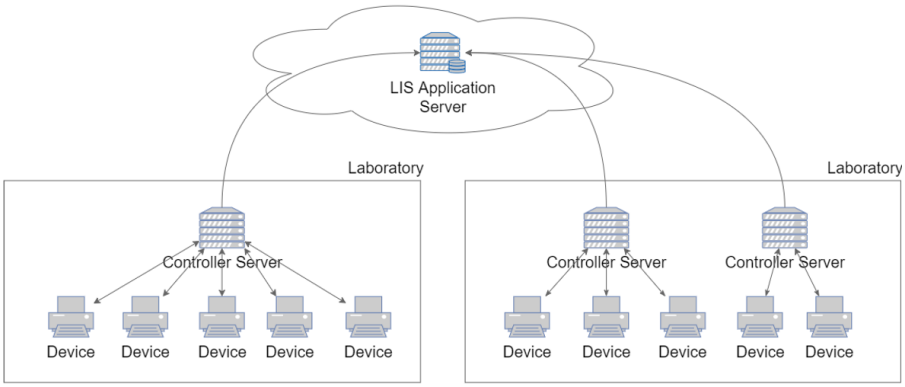


Fig. 1. D-LIS System Architecture

Figure 1 illustrates the D-LIS architecture. The number of devices represented is small, but in a real scenario there will be a higher number. The actual number of devices per site depends on the size of each laboratory. The number of devices that can be connected to a single controller server depends on the hardware characteristics of the server and the resources needed by each controller, which depends on the type of medical device under its control. This number is discussed in Section 3.

The D-LIS architecture described in this report is being implemented on an actual product - Clinidata® software [1]. Clinidata® is a web application designed for the global management of laboratories, blood banks and epidemiological surveillance. It was designed to be elastic, able to deal with large or small organizations such as global decentralized laboratories or large hospital centers. In order to integrate new devices in Clinidata®, several controllers were developed. Each controller was developed as a Java local application, which allows to achieve platform independence. The communication between the controllers and the LIS Application Server uses a set of web services exposed by the Clinidata® backend.

The connection between controllers and the LIS server is made through a non dedicated network (e.g., VPN over the Internet) where variations of latency and bandwidth may exist. This reason implies the need of creating some mechanisms to deal with faults and network failures.

In terms of device operation, typically laboratory devices use a message-acknowledgement approach, where strict timeouts are used. The classical approach for the processing of results sent by a device is the following: the controller periodically receives results from the device and forwards the data to the LIS server. Once data is received at the LIS, an acknowledgment (ACK) is generated and sent back to the device (through the controller) to confirm the data reception. The device waits for the ACK during a short period of time that typically varies between 3 and 20 seconds, depending on the brand and model.

When a D-LIS is considered, the network latency may compromise the response time and disrupt the normal procedure of the device. In order to deal with this, we propose a novel approach in which a local cache is added to the controller server. This allows the controller server to generate an ACK message for the device upon data is received by the controller server, optimizing the time required for each data transmission. This way, when a result is produced and sent to the controller, it is quickly cached, therefore a controller is able to send a faster confirmation message to the device. The results accumulated in the cache are periodically sent to the D-LIS by a parallel task that runs at the controller server. This task reads data from the cache, sends it to D-LIS and then deletes data from cache after being ensured that the data was received on D-LIS. In order to optimize the data transmission from controller server to the LIS server, we define a maximum of 500 records transferred at a time. This prevents heavy load on the web service at the LIS server, allowing it to deal with more controller servers simultaneously.

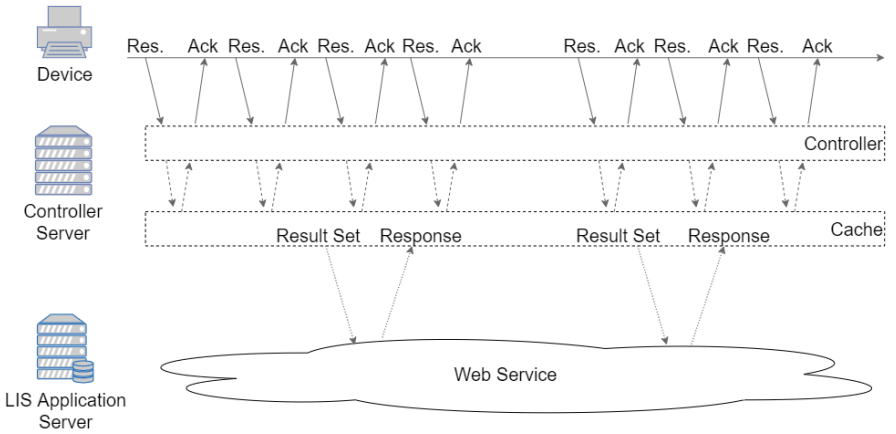


Fig. 2. Communication between components.

Figure 2 represents the interaction between all parts of the system. Here we can see the temporal-decoupling originated by using a cache between controller server and LIS Application Server.

In the D-LIS, there is a possibility of network failures occur for an undetermined time. If this happens, then the cache will continue to accumulate results allowing the devices to proceed with their normal operation. However, this leads to the impossibility of free up space in cache because results are not yet persistently stored in the LIS and cannot be deleted from the cache. To avoid memory exhausting in the controller server and to reinforce the persistence of the results, the cache mechanism uses a SQLite database to store results in disk. This ensures that no memory depletion occurs and at the same time increases the persistence of results by ensuring that in case of failures (e.g., power failures) result can be recovered from the local storage at the controller server.

Since each controller must be an independent process and due to SQLite descriptor limitation (only one writer at a time is supported), in our architecture there is a single file database for each controller. It adds some extra load for the controller server but once again enforces isolation between controllers.

3 Evaluation

The evaluation is the fundamental part of this work since it allows to conclude if the proposed architecture and the chosen technologies will achieve the real requirements needed in laboratory environments. An environmental simulation scenario close to the real laboratory environment was built. A set of controllers running in parallel on a test server (controller server) was used, where each controller is connected to one device. Each device was simulated through a simple program that connects to a given TCP/IP port (of the controller) and sends 1,000 messages. The messages are sent with an interval of 60 milliseconds and the size of each message is 1 KB. Each experiment ran during, on average, 40 minutes.

3.1 Metrics

The **first metric** considered in this work is the round-trip time (RTT) of a result message between the device and the LIS server, i.e., the time elapsed between the transmission of a result message by a device and the reception of the corresponding acknowledgment (ACK) by the same device. In this situation, a message must be generated and sent from the device, reach the LIS server, where it is stored and then an acknowledgement is generated and sent back to the device. It is important to quantify the time needed for that operation in order to ensure that it doesn't exceed the timeout value imposed by the devices with the most demanding real-time requirements.

In order to optimize the response time evaluated in the previous metric, the proposed architecture uses a cache mechanism to retain data quickly and sends an ACK after saving the result in the cache. The **second metric** considered in this work is the RTT of a result message between the device and the cache, i.e., the time elapsed between the transmission of a result message by a device to the controller server and the reception of the corresponding ACK by the same device when the cache mechanism is used, considering that the controller sends an ACK before ensuring that result was received by the LIS server.

It is also important that results must be fully available on the LIS server in a reasonable time. To assess whether this time does not exceed acceptable values, the **third metric** consists on the time elapsed between the result being sent by a device and the instant when the controller receives a confirmation of the LIS server that the result was stored in the central database.

In addition, we also collected memory (RAM) and storage (disk space) used by each controller. Despite being just a prototype, these values give an idea of what are the needed requirements for a controller server, depending on the number of controllers that are associated with.

Lastly, statistical information (average, standard deviation, maximum and minimum values) was computed for all metrics.

3.2 Results

All experiments described in this section consider the following setup: one controller server deployed on an Intel® Core™ i7-8700K @ 3.70 GHz with 32 GB of RAM memory; the system was evaluated for 1, 5, 10, 50, 100 and 250 controllers running in the same controller server; the controller server is connected to the LIS server through a stable local network connection; each experiment was run three times in order to collect statistics.

The first experiment performed considers the **first metric** - no cache mechanism. Table 1 shows the statistical information collected from this experiment.

Table 1. RTT (in ms) for device messages, no cache.

#Controller Instances	Average RTT	Std. Dev. RTT	Max. RTT
1	0.20	2.15	32
5	3.94	16.79	597
10	4.16	19.29	865
50	30.04	185.85	8 820
100	54.64	320.95	17 102

The values presented in Table 1 reveal that for a low number of controllers (up to 10 in parallel) the latency values are acceptable to guarantee the normal function of devices. However, we noticed an exponential increase in the latency values for 50 controllers. These values are unacceptable for many devices in which the maximum configurable timeout is less than 8 seconds. In this experiment, the results for 250 controllers were discarded because they were too high and thus no real system could operate with such performance.

The results displayed in Table 1 were more or less expected because all controllers are communicating directly with the LIS server without any kind of contention mechanism. Notice that these values would be higher in a real setup because they would be affected by the network latency of the internet connection between the controller server and the LIS server. In order to improve performance and fault-tolerance, the proposed architecture includes a cache mechanism in the controller server. In a second experiment we did the same test but now the cache is considered - **second metric**. Table 2 shows the results of this experiment.

Table 2. RTT (in ms) for device messages, using cache.

#Controller Instances	Average RTT	Std. Dev. RTT	Max. RTT
1	0.04	0.33	7
5	0.07	1.00	34
10	0.06	0.83	29
50	0.21	3.56	169
100	0.59	7.84	526
250	2.45	28.24	2 474

As expected, when the number of controllers increase, the RTT also increases. Note that each controller is totally independent. However, both average and maximum values still within the appropriate response time for most devices used in laboratories. In the case of 250 controllers the results show a significant increase in the RTT. This is a overload consequence on the hardware used to run the controller server in this experiment. The number of controllers that may run on a server depends on the server's specifications. Taking into account our hardware specifications, the configuration of 250 controllers in the same server should be avoided. Nevertheless, even in overload conditions, the maximum RTT was below 3 seconds, which complies with typical 3-20 seconds deadline range of most devices.

Since data must be delivered to the LIS server within a bounded latency, the next experiment reports the time taken since the production of a result by a device until it is stored in the LIS server - **third metric**. Table 3 shows the results of this experiment.

Table 3. Result latency (in ms) between device and LIS server, using cache.

#Controller Instances	Avg Latency	Std Dev Latency	Min Latency	Max Latency
1	1 009.07	575.47	11	2 004
5	999.60	578.48	5	2 006
10	1 006.65	576.71	4	2 005
50	1 003.14	577.09	2	2 041
100	1 002.31	577.33	2	2 154
250	1 063.33	642.50	4	4 553

From Table 3 we can conclude that results are available at the LIS server, on average, within 1 second. From the same table we can also see that the maximum can reach 4.5 seconds for 250 controllers. This time is high and occurs when results are stored in cache and the number of records in cache is greater than 500 results. In this situation results are only sent in the next iteration of the sending task that is used to transmit data from controller server to the LIS server. This is a particular scenario in our experiment. In a real situation, the controller server should be scaled up to deal with the number of controllers and its data rate, avoiding high latency as shown in this scenario.

Regarding memory usage (RAM), 50 MB are needed to run a controller. This value can increase to 60 MB when the controller is receiving and/or sending data. In terms of cache, the base SQLite metadata structure takes 16 KB of

storage. These values are concerned to a single controller. Since the system was designed to keep all controllers independent, the mentioned values are equal to all controllers. Based on these values we can estimate the minimum resources required for a controller server.

4 Conclusions

This practical experience report reflects the test-driven development of a distributed architecture for large groups of laboratories supported by many medical devices. Even through high load situations, the results show an acceptable response time, complying with the typical real-time requirements of laboratory devices. The results allow to conclude that the architecture offers a high degree of reliability in real scenarios. This is true when we compare the operation times of the D-LIS with and without using a cache. The cache plays an essential role not only to guarantee a response with predictable bounded time to the devices independently of the latency of the internet connection between the controller server and the LIS server, but also to provide persistence to the data produced by devices in an intermediate layer, ensuring that the devices keep operating even if network failures occur.

References

1. Clinidata®, https://www.maxdata.pt/, accessed: 2020-02-29
2. Altowaijri, S.M.: An architecture to improve the security of cloud computing in the healthcare sector. In: Smart Infrastructure and Applications, pp. 249–266. Springer (2020)
3. Clinical and Laboratory Standards Institute: Specification for transferring information between clinical laboratory instruments and information systems; approved standard - second edition (10 2004)
4. Donawa, A., Orukari, I., Baker, C.E.: Scaling blockchains to support electronic health records for hospital systems. In: 2019 IEEE 10th Annual Ubiquitous Computing, Electronics & Mobile Communication Conference (UEMCON). pp. 0550–0556. IEEE (2019)
5. Fouad, H., Mahmoud, N.M., El Issawi, M.S., Al-Feel, H.: Distributed and scalable computing framework for improving request processing of wearable iot assisted medical sensors on pervasive computing system. Computer Communications (2020)
6. Health Level Seven International: Hl7 version 2.5.1 (02 2007), http://www.hl7.eu/HL7v2x/v251/std251/hl7.html
7. Mathew, O.C., Dhanapal, R., Visalakshi, P., Parthiban, K., Karthik, S.: Distributed security model for remote healthcare (dsm-rh) services in internet of things environment. Journal of Medical Imaging and Health Informatics **10**(1), 185–193 (2020)
8. Mutlag, A.A., Ghani, M.K.A., Arunkumar, N.a., Mohammed, M.A., Mohd, O.: Enabling technologies for fog computing in healthcare iot systems. Future Generation Computer Systems **90**, 62–78 (2019)
9. Thompson, D., Wright, K.: Developing a unified patient-record: a practical guide. CRC Press (2018)

Abreviaturas

ASTM American Society for Testing and Materials. 10

FCUL Faculdade de Ciências da Universidade de Lisboa. i, 1

GWT Google Web Toolkit. 16, 17

HL7 Health Level 7. 10

HQL *Hibernate Query Language*. 17, 30

HTML *Hypertext Markup Language*. 16

LIS Laboratory Information System. 1, 2, 8–10

PEI Projeto em Engenharia Informática. i, 1, 2, 7, 18, 19, 31, 35, 38

SGBD Sistema de Gestão de Base de Dados. 20

SOAP *Simple Object Access Protocol*. 30, 31

SQL *Structured Query Language*. 20

SSL *Secure Sockets Layer*. 38

XML *Extensible Markup Language*. 17–19

Bibliografia

[1] Maxdata Software, SA. https://www.maxdata.pt/. Acedido em: 25/08/2020.

[2] Apache. Apache Commons Deamon. https://commons.apache.org/proper/commons-daemon/. Acedido em: 25/08/2020.

[3] Apache. Apache Maven. https://maven.apache.org/. Acedido em: 25/08/2020.

[4] com0com. Null-modem emulator com0com. http://com0com.sourceforge.net/. Acedido em: 25/08/2020.

[5] Confidentia. Software Appolo - Confidentia. http://www.confidentia.pt/pt/software-appolo. Acedido em: 22/11/2019.

[6] 3 fases dos exames laboratoriais. http://www.qualichart.com.br/blog/3-fases-dos-exames-laboratoriais-2-fase-analitica/. Acedido em: 22/11/2019.

[7] Git. Git. https://git-scm.com/. Acedido em: 25/08/2020.

[8] GitLab. Gitlab. https://about.gitlab.com/. Acedido em: 25/08/2020.

[9] Google. [GWT]. http://www.gwtproject.org/. Acedido em: 25/08/2020.

[10] Hibernate. Hibernate. https://hibernate.org/. Acedido em: 25/08/2020.

[11] ICARCV. Icarcv 2020. https://www.icarcv.sg/. Acedido em: 25/08/2020.

[12] Liquibase. Liquibase | Open Source Version Control for Your Database. https://hibernate.org/. Acedido em: 25/08/2020.

[13] LumenLearning. Measuring drug susceptibility. https://courses.lumenlearning.com/boundless-microbiology/chapter/measuring-drug-susceptibility/. Acedido em: 25/08/2020.

[14] S.A Maxdata Software. Documentos Internos SGI (não publicado).

[15] Roger S. Pressman. Software engineering: a practitioner's approach, 5th edition. 2001.

[16] Maxdata Software S.A. Clinidata XXI - Patologia Clinica. https://www.maxdata.pt/pt/produto/3/clinidataxxi-patologia-clinica/, 10 Agosto 2018. Acedido em: 22/11/2019.

[17] SafeComp. Safecomp 2020. http://safecomp2020.di.fc.ul.pt/. Acedido em: 25/08/2020.

[18] AGG Software. Com Port Data Emulator. https://www.aggsoft.com/com-port-emulator.htm. Acedido em: 25/08/2020.

[19] Orchard Software. Harvest - Orchard Software. https://www.orchardsoft.com/orchard-harvest.html. Acedido em: 22/11/2019.

[20] SQLite. https://www.sqlite.org/. Acedido em: 25/08/2020.

[21] SQLite Tutorial. Connect to sqlite database using sqlite jdbc driver. https://www.sqlitetutorial.net/sqlite-java/sqlite-jdbc-driver/. Acedido em: 25/08/2020.

[22] VMware. Spring. https://spring.io/. Acedido em: 25/08/2020.